МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

С.Р. Михайлов

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ
Лабораторний практикум

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 171 «Електроніка»,
спеціалізацією «Електронні прилади та пристрої»*

Київ
КПІ ім. Ігоря Сікорського
2019

Рецензент *Тодоренко В.А..,* канд. техн. наук, доцент, доцент кафедри промислової електроніки КПІ ім. Ігоря Сікорського

Відповідальний редактор *Писаренко Л.Д.*, д-р техн. наук, професор

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № від р.) за поданням Вченої ради факультету електроніки (протокол № 03/2019 від 25.03.2019 р.)

Електронне мережне навчальне видання

Михайлов Сергій Ростиславович, канд. техн. наук, доц.

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ
Лабораторний практикум

Основи мікропроцесорної техніки: Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 171 «Електроніка», спеціалізації «Електронні прилади та пристрої» / С.Р. Михайлов; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,75 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2019. – 59 с.

Метою лабораторного практикуму з дисципліни «Основи мікропроцесорної техніки» є закріплення та поглиблення теоретичних знань по архітектурі, принципах побудови, функціонування та програмування сучасних 8-розрядних мікроконтролерів сімейства MCS-51 з CISC-архітектурою та мікроконтролерів AVR фірми Atmel з RISC-архітектурою, а також набуття практичних навичок для роботи з такими мікроконтролерами.

Навчальний посібник призначений для студентів, які навчаються за спеціальністю 171 «Електроніка», спеціалізацією «Електронні прилади та пристрої», може бути корисним студентам інших спеціальностей та спеціалізацій.

ЗМІСТ

ВСТУП

Однокристальний мікроконтролер – пристрій, конструктивно виконаний у вигляді великої інтегральної схеми, що містить усі компоненти мікропроцесорної системи: процесор, пам'ять даних, пам'ять програм, паралельні і послідовні порти введення–виведення та інші вузли. Однокристальні мікроконтролери (МК) є зручним інструментом для створення сучасних вбудованих пристроїв керування різним обладнанням в таких галузях, як побутова техніка, автомобільна електроніка, медична електроніка, верстатобудування, мобільний зв'язок тощо. Кількість користувачів мікроконтролерів значно перевищує кількість користувачів однокристальних мікропроцесорів (МП). Відповідно об'єми виробництва мікроконтролерів у декілька разів більше, ніж однокристальних МП.

У теперішній час різними фірмами випускаються 8-, 16- та 32-розрядні МК з об'ємом пам'яті програм до десятків Кбайт, невеликим об'ємом пам'яті даних та набором таких інтерфейсних і периферійних схем, як паралельні та послідовні порти введення–виведення, таймери, аналого-цифрові та цифро-аналогові перетворювачі, широтно-імпульсні модулятори тощо.

Незважаючи на появу нових 16- та 32-розрядних МК, найбільш розповсюдженими є 8-розрядні, які займають приблизно половину об'єму світового ринку МК. Причиною цього є те, що велика кількість задач керування та регулювання вирішується саме за допомогою 8-розрядних МК.

Тому запропонований навчальний посібник описує лабораторні роботи саме по сучасним 8-розрядним мікроконтролерам сімейства MCS-51 з CISC-архітектурою та мікроконтролеам AVR фірми Atmel з RISC-архітектурою.

Метою лабораторних робіт з дисципліни «Основи мікропроцесорної техніки» є закріплення та поглиблення теоретичних знань по архітектурі, принципах побудови, функціонування та програмування сучасних 8-розрядних мікроконтролерів. а також набуття практичних навичок для роботи з такими мікроконтролерами.

ЛАБОРАТОРНА РОБОТА № 1

Дослідження арифметичних та логічних операцій мікроконтролерів сімейства MCS-51.

1. Мета роботи:

Ознайомитися з основними арифметичними та логічними операціями мікроконтролерів сімейства MCS-51.

2. Програма роботи:

2.1. Ознайомитися з лабораторним стендом “EV8031/AVR”.

2.2. Ознайомитися з архітектурою та системою команд мікроконтролерів сімейства MCS-51.

2.3. Скласти програму на мові Асемблер для обчислення виразу (табл. 1). Результат обчислення виразу відобразити на чотирьохрозрядному семисегментному світлодіодному індикаторі HL1 лабораторного стенда (адреса двох молодших розрядів індикатора – 0B000H, адреса двох старших розрядів – 0A000H).

2.4. На персональному комп'ютері завантажити текстовий редактор (Total Commander).

2.5. У текстовому редакторі набрати текст програми в мнемокодах мови Ассемблер для MCS-51.

2.6. Зберегти набраний файл із розширенням *.ASM.

2.7. Відкомпілювати набрану програму відповідними засобами (наприклад, компілятором ASM51).Для компіляції у командному рядку Total Commander набрати: ASM51.EXE NAME.ASM, де NAME - ім’я збереженого файлу (файли ASM51 та NAME повинні знаходитися в одній директорії).

2.8. Можливі помилки в програмі можна переглянути в однойменному файлі з розширенням *.LST.

2.9. Після усунення всіх помилок, дані файлу з розширенням *.HEX програмою EVAL32.EXE необхідно перенести в стенд. Для цього у командному рядку Total Commander набрати: EVAL32.EXE –hs –com 1 9600 NAME.HEX.

2.10. Вивід на екран підказки про параметри програми EVAL32.EXE,

здійснюється запуском EVAL32.EXE.

2.11.При передаванні даних з персонального комп'ютера в лабораторний стенд на екрані монітора відображаються дані, що передаються. Ці ж дані відображаються на індикаторі стенда HL1. Горить світлодіод HL9. Після передавання останнього байта завантажена програма запускається автоматично.

2.12.При необхідності перезапуску завантаженої в стенд програми натиснути кнопку SW1.

2.13.Зупинка завантаженої програми та перехід у режим очікування на прийом даних з персонального комп'ютера можливо натисканням кнопки SW2. При цьому гасне світлодіод HL9. Запис нової програми можливий в будь-який момент часу роботи завантаженої програми.

Вихідні дані для виконання лабораторної роботи Таблиця 1

Варіант	1	2	3
Вираз	(64:4+13·9)⊕ 65	(122-58):8 v 17·12	(18·14)⊕ (240:15)

4. Зміст звіту

- Титульний лист з відомостями про назву роботи і склад бригади.
- Текст програми з коментарями.

5. Контрольні запитання

- Назвіть групи команд мікроконтролера сімейства MCS-51.
- Назвіть основні арифметичні та логічні команди.
- Які методи адресації існують в мові Асемблера мікроконтролера сімейства MCS-51? Наведіть приклади.
- З даними яких форматів оперують команди мікроконтролера MCS-51?
- Назвіть основні складові мікроконтролера MCS-51.

6. Опис лабораторного стенда

Лабораторний стенд “EV8031/AVR” - програмно-апаратний комплекс,

орієнтований для застосування в навчальних цілях (дисципліна ”Мікропроцесорна техніка”), а також як засіб розробки програмного забезпечення мікроконтролерів серій MSC-51 та AVR. Зв'язок лабораторного стенда “EV8031” з ПК здійснюється через COM-порт.

Технічні характеристики стенда

- Однокристальні мікроконтролери AT89C51, AT89C52, AT90S8515 (ATmega8515) (корпус DIP-40);
- Пам'ять програм - 16 Кбайт;
- Пам'ять даних - 16 Кбайт;
- Послідовна EEPROM пам'ять, 256 байт (AT24C02);
- Два послідовних канали передачі даних RS232;
- Клавіатура 4x3
- Статична 4-розрядна семисегментна світлодіодна індикація;
- Цифро-аналоговый й аналого-цифровий перетворювач (плата розширення);
- Генератор з фіксованою частотою генерації – (близько 10 кГц), генератор зі змінюваною частотою генерації (від 1 кГц до 50 кГц) (плата розширення);
- Динамічна 4-х розрядна семисегментна індикація (плата розширення);
- Пристрій дискретного введення інформації: 2 кнопки;
- Статична світлодіодна індикація, 8 шт.;
- Знакосинтезуючий світлодіодний індикатор 5x7 (плата розширення).

Пам'ять ОЗП обсягом 32К поділяється на дві частини по 16К (рис.1). Одна частина для пам'яті програм, інша - для пам'яті даних. У режимі завантаження вся пам'ять 32К відображається в адресний простір, як пам'ять даних.

При надходженні даних з послідовного порту персонального комп'ютера в послідовний порт (рознімання X11) стенда, МК записує їх в ОЗП, яке відведене під пам'ять програм. Сигнали керування - PME, WR, RD, ALE, сформовані процесором і необхідні для звертання до пам'яті даних надходять через системний контролер. Після прийняття останнього байта завантажник формує сигнал запуску програми

шляхом запису керуючого коду в системний контролер.

Кнопка SW2 необхідна для формування сигналу скидання на вході RESET процесора, тобто переходу стенда в режим завантаження й очікування прийому даних з послідовного порту. При цьому МК готовий приймати дані в пам’ять даних. Кнопка SW1 необхідна для перезапуску завантаженої із ПК програми, що перебуває в пам'яті програм.

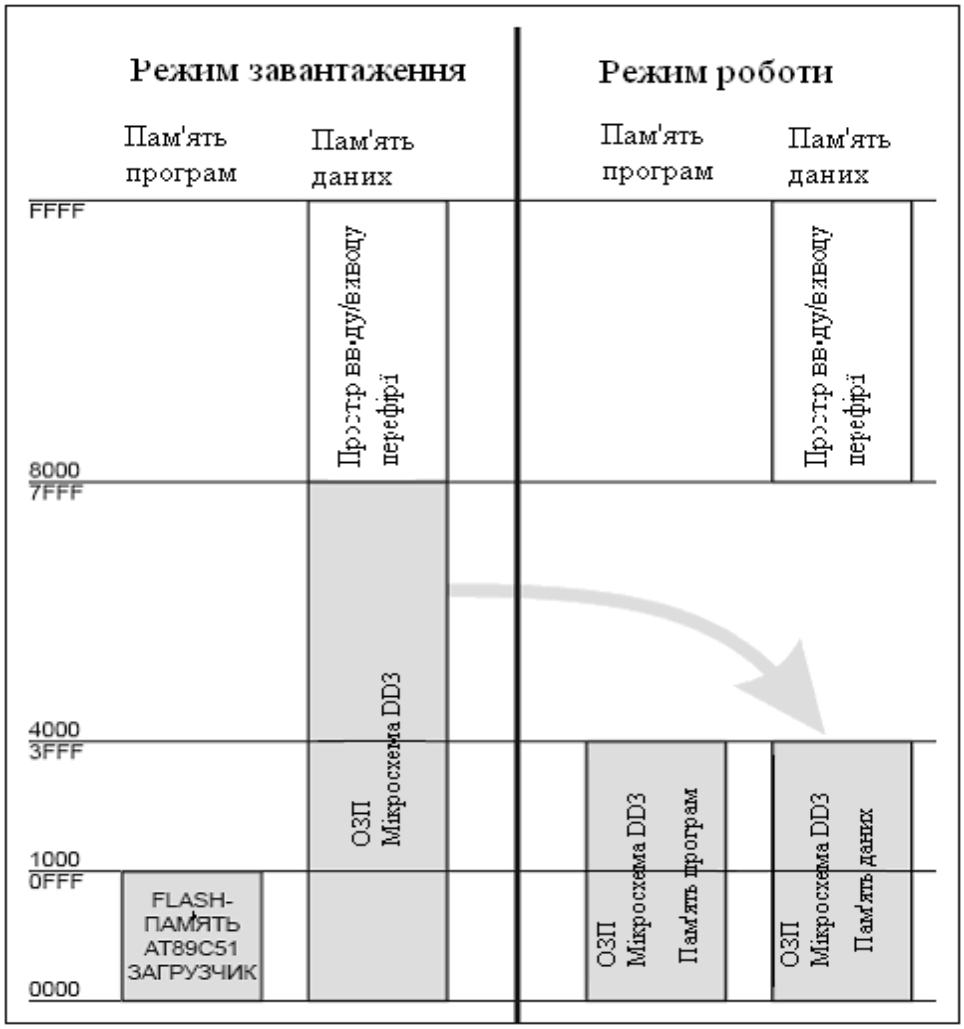


Рис. 1. Розподіл пам'яті стенда.

Адресація (звернення) процесора до периферійних пристроїв стенда реалізована як адресація до комірок пам'яті в адресному просторі від 8000H до FFFFH. Сигнали вибірки периферійних пристроїв формуються дешифратором адреси усередині мікросхеми системного контролера DD4.

Структура стенда (рис.2) реалізована на програмувальній логічній мікросхемі

EPM7128STC100 (DD4). Системний контролер керує режимами роботи, виробляє керуючі сигнали на ОЗП, регістри, динамічний світлодіодний індикатор, клавіатуру.

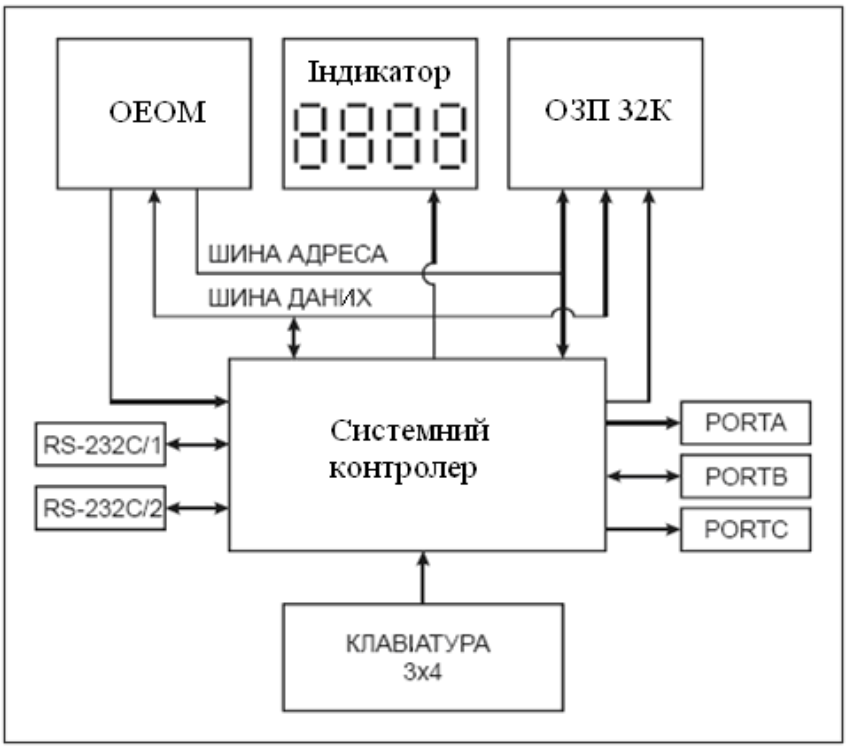


Рис. 2. Структурна схема стенда

Світлодіодний індикатор.

Чотирьохрозрядный семисегментний світлодіодний індикатор підключений до системного контролера, що автоматично виконує динамічну регенерацію й декодування двійкового коду в код семисегментного індикатора. Індикатор працює завжди, відразу після подачі живлення. Контролер індикатора містить два восьмирозрядных регістри, вміст яких відображається на індикаторі. Вміст регістра з адресою 0A000 відображається на двох лівих розрядах, вміст регістра с адресою 0B001(0x000) - на двох правих розрядах у шістнадцятковій формі.

Матрична клавіатура.

Стан стовпця матриці клавіатури зчитується з комірки з базовою адресою 0x9000, біти 3..0. Відповідний стовпець вибирається нулем у розрядах адреси

A2..A0. Тобто, адреса 0x9006 вибирає перший стовпець, адреса 0x9005 - другий стовпець, адреса 0x9003 - третій стовпець. Ознака натиснутої кнопки зчитується як нуль у відповідному розряді.

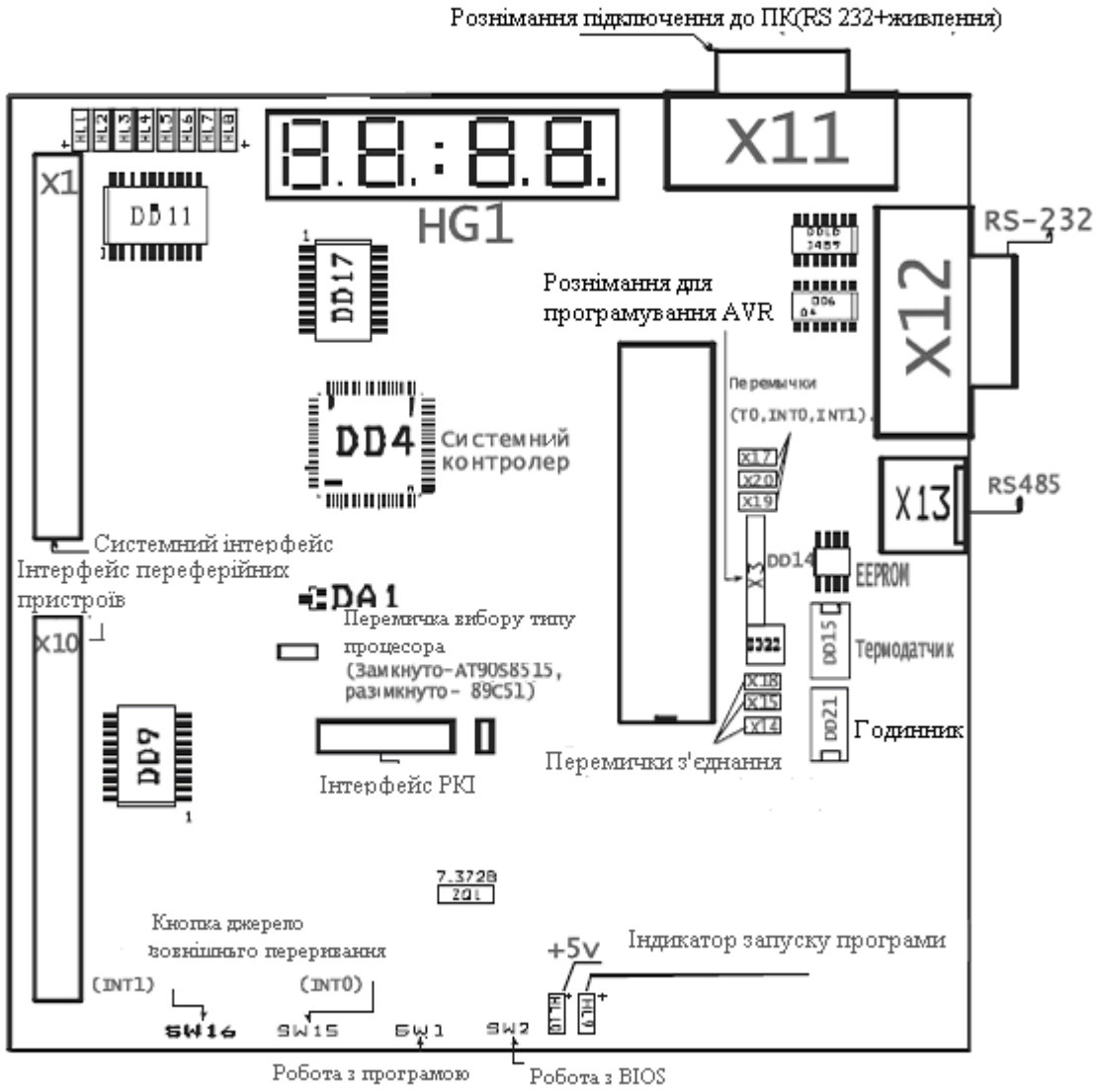


Рис. 3. Схема розташування елементів стенда.

X1 - Системний інтерфейс із повним адресним простором;

X10 - Інтерфейс розширення для підключення зовнішніх пристроїв з використанням паралельного інтерфейсу;

X11 - Інтерфейс послідовного порту COM1 для зв'язку стенда із PC;

X12 - Інтерфейс послідовного порту COM2 для зв'язку стенда з іншими пристроями, що мають стандартний порт RS232C;

X3 - Інтерфейс програмування AVR.

7. Структура мікроконтролерів сімейства MCS-51

В дійсний час найбільш розповсюдженим серед усіх 8-розрядних мікроконтролерів (МК) є сімейство MCS-51 з CISC-архітектурою фірми Intel. Воно одержало свою назву від першого представника цього сімейства - МК i8051. Вдалий набір периферійних пристроїв, можливість гнучкого вибору зовнішньої або внутрішньої пам'яті програм і невисока вартість забезпечили цьому МК успіх на ринку. У результаті на сьогоднішній день існує більш 200 модифікацій МК сімейства i8051, що випускаються майже 20-ю компаніями.

Усі МК сімейства MCS-51 мають загальну систему команд. Наявність додаткового устаткування впливає тільки на кількість регістрів спеціального призначення. Вітчизняний аналог МК i8051 – К1816ВЕ51.

Базова модель сімейства MCS-51 МК i8051 містить :

- вбудований тактовий генератор;
- внутрішню (резидентну) пам'ять програм ПЗП об’ємом 4 КБайт;
- внутрішню (резидентну) пам'ять даних ОЗП об’ємом 128 байт;
- можливість адресації 64 КБайт пам'яті програм та 64 КБайт пам'яті даних;
- 32 двонапрямлені та індивідуально адресовані лінії введення-виведення (порти P0-P3);
- послідовний порт введення/виведення*;*
- два 16-розрядні багатофункціональні таймери-лічильники;
- дві лінії запитів на переривання від зовнішніх пристроїв.

Структурна схема МК i8051 наведена на рис. 4. Розглянемо склад схеми та призначення її елементів.

Мікроконтролер містить 8-розрядний центральний процесор ЦП, резидентну пам'ять програм (ПЗП) об’ємом 4 КБайт, резидентну пам'ять даних (ОЗП) об’ємом 128 байт, чотири 8-розрядних програмованих порти введення-виведення P0-P3, блок керування БК і блок переривань, таймерів та послідовного порту.

Центральний процесор ЦП містить 8-розрядний арифметико-логічний

пристрій АЛП зі схемою десяткової корекції СДК, два акумулятори A і B, регістр слова стану процесора PSW *(Processor State Word)* та програмно-недоступні буферні регістри T1 і T2, які виконують функції розподілу вхідних та вихідних даних АЛП. Центральний процесор виконує операції додавання, віднімання, множення, ділення, логічні операції І, АБО, ВИКЛЮЧАЛЬНЕ АБО, операції зсуву, інверсії та скидання. Він оперує з такими типами змінних: булевими (1 біт), цифровими (4 біт), байтовими (8 біт) та адресними (16 біт). Особливістю МК є великий набір операцій з бітами: окремі біти даних можуть бути встановлені, скинуті, інвертовані, перевірені, передані.

Акумулятор А є джерелом одного з операндів і місцем розміщення результату виконання багатьох команд. Акумулятор В використовується як акумулятор лише у командах множення і ділення, а в інших випадках - як регістр загального призначення.

Регістр слова стану процесора PSW зберігає інформацію про стан АЛП у процесі виконання програми і має формат, наведений у табл. 2. Прапорець перенесення (позики) С встановлюється у разі виходу результату додавання (віднімання) беззнакових операндів за межу діапазону. Прапорець додаткового перенесення (позики) AC встановлюється у разі перенесення з молодшої тетради у старшу (з розряду D3 у розряд D4). Прапорець OV встановлюється у командах додавання і віднімання, якщо результат перевищує ємність 7 біт і старший біт не може бути інтерпретований як знаковий у командах ділення відбувається скидання OV, а під час ділення на нуль він знову встановлюється. У командах множення OV набуває значення логічної одиниці, якщо результат перевищує 0FFH. Прапорець P є доповненням вмісту акумулятора А до парності, тобто 9-розрядне слово, що складається з 8 біт акумулятора А та біта P, має завжди парну кількість одиниць.

Блок керування БК складається з генератора тактових імпульсів Г, схеми керування і синхронізації СКС та програмно-недоступного регістра команд (РК). Код команди, зчитаної з резидентної пам’яті програм, запам’ятовується у 8-розрядному РК і надходить на дешифратор команд, який входить до складу СКС. На підставі декодованого коду команди, зовнішніх керуючих сигналів та сигналів

синхронізації блок керування БК виробляє внутрішні сигнали керування блоками МК.

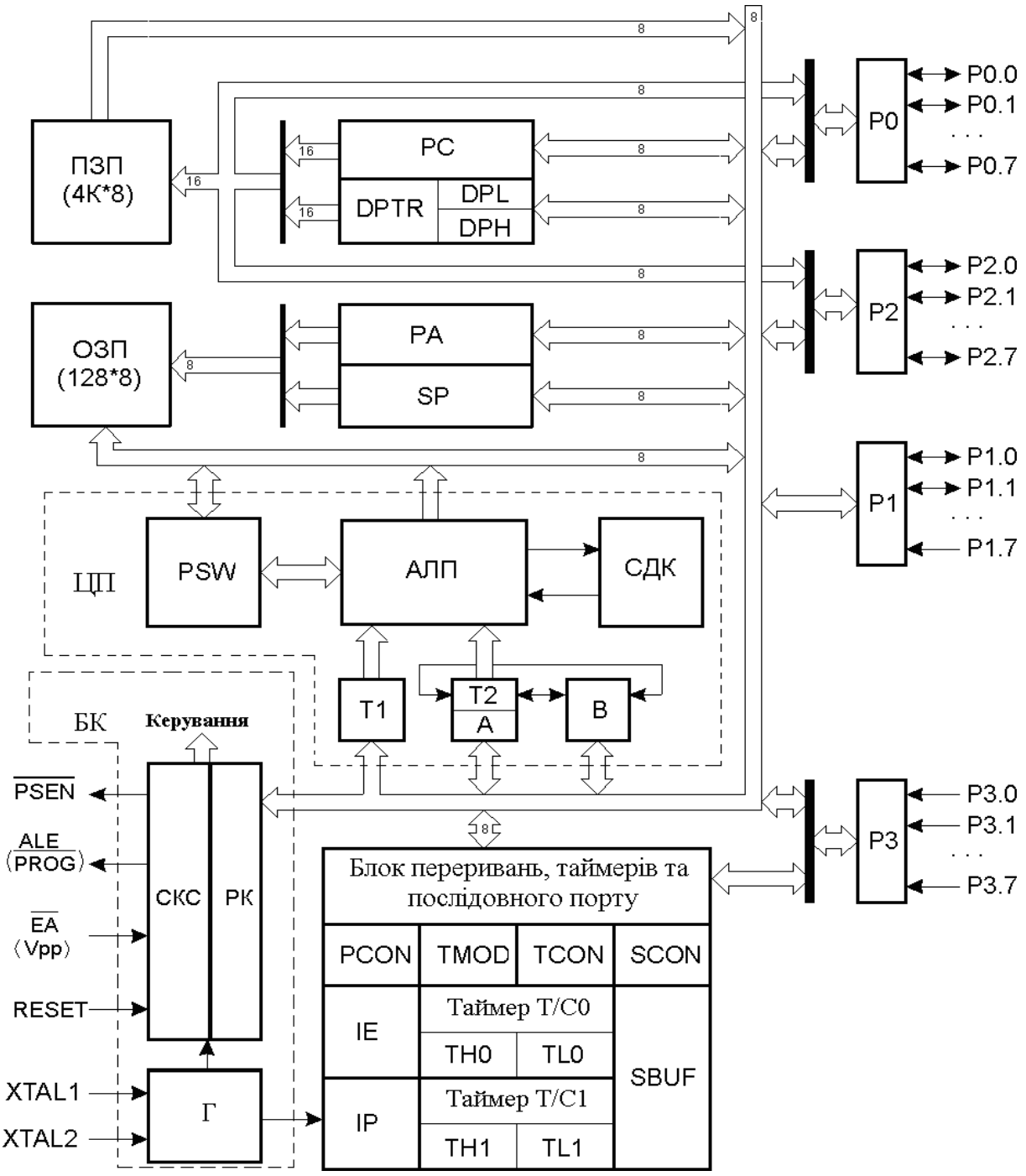


Рис. 4. Структурна схема мікроконтролера i8051

Постійний запам'ятовувальний пристрій ПЗП, або резидентна пам'ять програм (РПП) МК i8051 має інформаційну ємність 4 Кбайт і виконана у вигляді ПЗП програмною маскою. ПЗП має 16-розрядну адресну шину, що дає змогу розширити пам'ять до 64 Кбайт через під'єднання зовнішніх ВІС. Адреса визначається вмістом лічильника команд PC *{Program Counter)* або вмістом регістра-покажчика даних DPTR

(Data Pointer Register). Регістр DPTR використовується за непрямих переходів у програмі або під час адресації таблиць, або як один 16-розрядний регістр, або як два незалежних 8-розрядних регістри DPH і DPL.

Формат слова стану PSW Таблиця 2

Біт	Позначення	Призначення	Доступ до біта
7	C	Прапорець перенесення	А або П
6	AC	Прапорець додаткового перенесення	А або П
5	F0	Прапорець користувача	П
4	RS1	Покажчик банку робочих регістрів: 00 – банк 0; 10 – банк 1; 01 – банк 2; 11 – банк 3	П
3	RS0		
2	OV	Прапорець переповнення	П
1	–	Резервний прапорець	П
0	P	Біт парності	А або П

Примітка: У таблиці використано такі позначення: А – біт встановлюється апаратно; П - біт встановлюється програмно.

Розподіл адресного простору ПЗП показано на рис. 5. Молодші адреси ПЗП відводяться для оброблення переривань і початку роботи МК після скидання.

Оперативний запам'ятовувальний пристрій ОЗП, або резидентна пам'ять даних (РПД) складається з двох областей (рис. 6). Перша область - ОЗП даних з інформаційною ємністю 128 х 8 біт з адресами 0 — 7FH, друга область - регістри спеціальних функцій *(SFR — Special Function Registers)* з адресами 80H — FFH.

Резидентна пам'ять даних адресується 8-розрядними регістром адреси (РА) або покажчиком стеку (SP) (див. рис. 4). Регістр адреси є програмно-недоступним регістром, у який завантажується адреса комірки ОЗП під час виконання команд. Регістр SP призначений для адресації стеку, який є частиною РПД. Вміст SP інкрементується перед запам'ятовуванням даних у стеку за командами PUSH і CALL та декрементується за командами POP і RET. У процесі ініціалізації МК після надходження сигналу скидання (RESET) у SP автоматично завантажується код 07H. Отже, якщо програма не перевизначає стек, то перший байт даних у стеку

розміщується у комірці РПД за адресою 08H.

Резидентна пам'ять даних, так само як і РПП, може бути розширена до 64 Кбайт за допомогою зовнішніх ВІС.

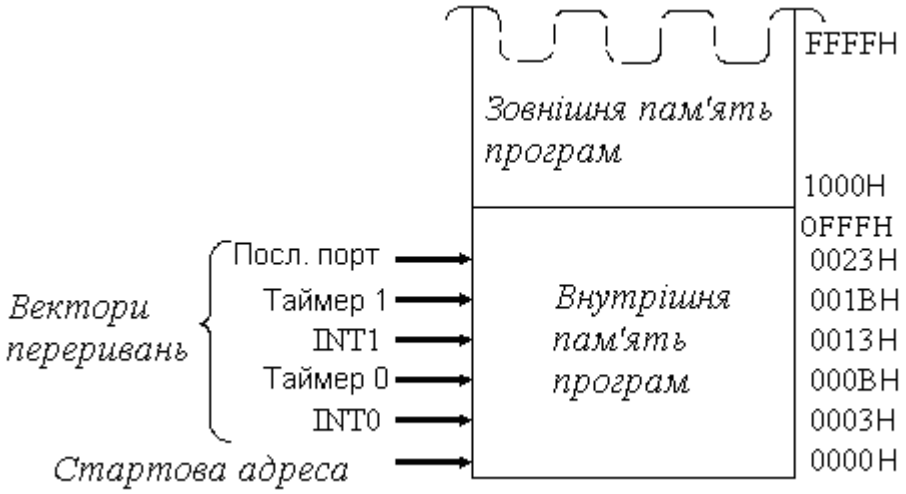
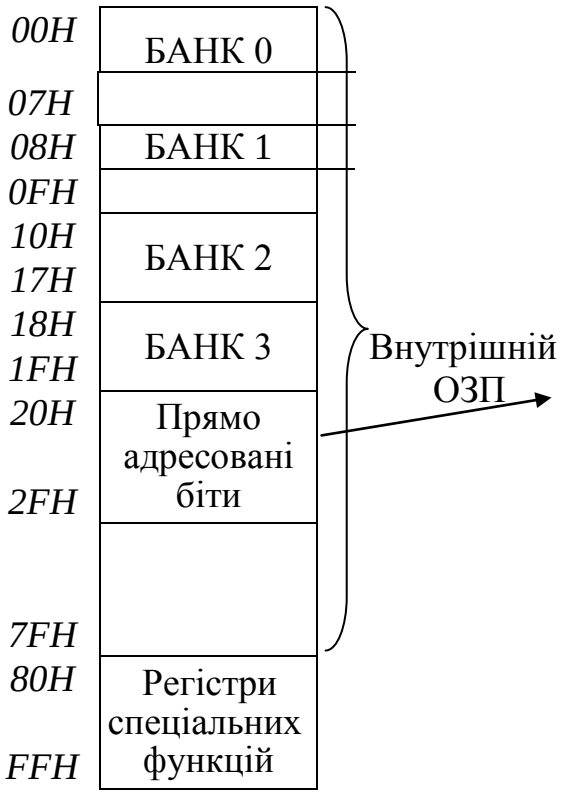


Рис. 5. Розподіл адресного простору РПП

Блок переривань, таймерів та послідовного порту (рис. 4) містить два таймери-лічильники T/C0 та T/C1, регістр режимів таймерів-лічильників TMOD *(Timer/Counter Mode Control Register)*, регістр керування таймерами-лічильниками TCON *(Timer/Counter Control Register)*, буфер послідовного порту SBUF *(Serial Data Buffer)*, регістр керування послідовним портом SCON *(Serial Port Control Register),* регістр дозволу переривань IE *(Interrupt Enable Register)*, регістр пріоритетів переривань IP *(Interrupt Priority Control Register)* та регістр керування енергоспоживанням PCON *(Power Control Register)*.

Таймери-лічильники містять два 16-розрядних регістра T/C0 та T/C1. Кожний з цих регістрів складається з двох 8-розрядних регістрів – TH0, TL0 і TH1, TL1 відповідно.

Всі регістри блоку переривань, таймерів та послідовного порту, а також регістри DPTR, SP, PSW, акумулятори A і B, регістри портів P0 – P3 відносяться до регістрів спеціальних функцій SFR (*Special Function Registers*), які розташовані у РПД за адресами 80H-FFH (рис. 6). Перелік регістрів спеціальних функцій наведений у таблиці 3.



Прямо адресовані біти

	D7							D1
20H	07	06	05	04	03	02	01	00
21H	0F	0E	0D	0C	0B	0A	09	08
22H	17	16	15	14	13	12	11	10
23H	1F	1E	1D	1C	1B	1A	19	18
24H	27	26	25	24	23	22	21	20
25H	2F	2E	2D	2C	2B	2A	29	28
26H	37	36	35	34	33	32	31	30
27H	3F	3E	3D	3C	3B	3A	39	38
28H	47	46	45	44	43	42	41	40
29H	4F	4E	4D	4C	4B	4A	49	48
2AH	57	56	55	54	53	52	51	50
2BH	5F	5E	5D	5C	5B	5A	59	58
2CH	67	66	65	64	63	62	61	60
2DH	6F	6E	6D	6C	6B	6A	69	68
2EH	77	76	75	74	73	72	71	70
2FH	7F	7E	7D	7C	7B	7A	79	78

Рис. 6. Резидентна пам’ять даних

Регістри спеціальних функцій Таблиця 3

Адреса	Позначення	Найменування
0E0H	ACC*	Акумулятор A
0F0H	B	Акумулятор B
0D0H	PSW*	Слово стану процесора
080H	P0*	Порт 0
090H	P1*	Порт 1
0A0H	P2*	Порт 2
0B0H	P3*	Порт 3
081H	SP	Регістр покажчик стеку
083H	DPH	Старший байт регістра-покажчика даних DPTR
082H	DPL	Молодший байт регістра-покажчика даних DPTR
08CH	TH0	Старший байт таймера 0
08AH	TL0	Молодший байт таймера 0
08DH	TH1	Старший байт таймера 1
08BH	TL1	Молодший байт таймера 1
089H	TMOD	Регістр режимів таймерів-лічильників
088H	TCON*	Регістр керування-статусу таймерів

Продовж. табл. 3

0B8H	IP*	Регістр пріоритетів переривань
0A8H	IE*	Регістр дозволу переривань
087H	PCON	Регістр керування енергоспоживанням
098H	SCON*	Регістр керування послідовним портом
099H	SBUF	Буфер послідовного порту

Примітка. Позначені регістри допускають адресацію окремих бітів.

8. Система команд мікроконтролерів сімейства MCS-51

Система команд МК i8051 містить 111 команд, які поділяють на п'ять груп: команди пересилання; арифметичні команди; логічні команди; команди передавання керування; команди операцій з бітами. Команди оперують з одно-, чотири-, вісьми- та 16-бітними словами.

Більшість команд (94) мають формат 1 - 2 байт і виконуються за один-два цикли (за тактової частоти 12 МГц тривалість циклу дорівнює 1 мкс).

Систему команд МК i8051 наведено у табл. 4, де використано такі позначення:

Rn – один з регістрів R0-R7;

Ri – регістр R0 або R1;

ad – байт з РПД за прямою адресацією (у команді вказується або адреса байта – 00-FF, або позначення одного з регістрів спеціальних функцій SFR, наприклад, TH0, P1, TCON, SBUF);

D8 – безпосередні 8-розрядні дані (перед числовими значеннями ставлять знак #, наприклад #34);

D16 – безпосередні 16-розрядні дані;

add – байт-приймач з прямою адресацією;

ads – байт-джерело з прямою адресацією;

ad16 – 16-розрядна адреса;

ad11 – 11-розрядна адреса;

ad8 – 8-розрядна адреса-зміщення;

РПД – резидентна пам'ять даних; ЗПД – зовнішня пам'ять даних; ПП – пам'ять програм;

bit – біт з прямою адресацією (у команді вказується або адреса біта – 00-FF, або розташування його у регістр спеціальних функцій, наприклад ACC.0 - нульовий біт акумулятора, або позначення біта у регістрі спеціальних функцій, наприклад EA - біт загального дозволу переривань).

Мікропроцесори MCS-51 допускають наступні методи адресації: **безпосередню, пряму, регістрову, непряму регістрову, індексну, символічну.**

Безпосередня адресація дозволяє занести на адресу призначення константу, що безпосередньо вказана в команді, наприклад:

MOV A, #100; в акумулятор записується десяткове число 100.

Пряма адресація використовується для звертання до SFR-області та внутрішніх регістрів RAM. Можлива як бітова так і байтова адресація.

Допускається пряма байтова адресація до внутрішніх регістрів RAM з номерами 0 - 127, наприклад:

MOV A, 25H ; в акумулятор записується вміст регістра з адресою 25H.

Для бітової адресації регістрів внутрішньої RAM доступні регістри з номерами 20H - 2FH. Біти цих регістрів послідовно нумеровані та мають адреси 0 - 127. Сукупність цих біт складає бітову RAM - область мікропроцесора. В командах з бітовою прямою адресацією можливе використання як прямих адрес бітової, так і байтової RAM-областей.

Наприклад, тотожними є наступні команди:

SETB 0; встановлення нульового біта бітової RAM - області в "1"

SETB 20H. 1; встановлення нульового біта регістра 20H байтової RAM - області в "1", тому що в обох випадках адресується один і той же біт внутрішньої RAM-області.

Прямі адреси регістрів SFR-області мають номера 128 - 255, які не перетинаються з адресами регістрів RAM. Старший біт байта коду прямої адреси визначає, до чого виконується звернення: до RAM чи до SFR області мікропроцесора.

Для бітової адресації придатні наступні регістри SFR: P0, PI, P2, P3, TCON, SCON, IE, IP, PSW, ACC, B.

Регістрова адресація використовується для звертання до обраного банку регістрів загального призначення R0 - R7, регістрів A, B, AB, DPTR, бітового акумулятора C. її використання дозволяє отримати двобайтовий код команди, еквівалентний по результатам дії трибайтовому коду, що утворюється при використанні прямої адресації. Економія виникає за рахунок того, що в коді операції розміщено також адресу одного з операндів - регістра. Наприклад, передачу даних з регістру RAM за номером 44H в регістр OH (або R0) можна реалізувати наступними способами:

MOV OH, 44H; пряма адресація - запис команди займає 3 байти в ROM

MOV R0, 44H; регістрова адресація - запис команди займає 2 байти.

Такий метод адресації дозволяє більш економно використовувати пам'ять програм.

Непряма регістрова адресація використовується для звертання:

- до регістрів внутрішньої RAM з використанням регістрів покажчиків RO, R1 обраного банку регістрів загального призначення. В мікропроцесору всього 8 регістрів такого типу і тому їх необхідно використовувати досить обачливо. Регістр покажчик вказує на номер регістру RAM, що адресується, на приклад:

MOV R0, #2; в регістр покажчик R0 записана адреса регістру до якого буде записане число MOV A, #10; запис числа в акумулятор.

MOV @R0, A; передача числа з акумулятора на вказану адресу.

- до регістрів зовнішньої пам'яті даних. Можливе використання як бай тового так і двобайтового покажчиків. Якщо використовується байтовий покажчик (регістри RO, R1), то обирається комірка зовнішньої пам'яті даних з сторінки - блоку в 256 байт. Номер сторінки попередньо вказується за допомогою порту P2, наприклад,

MOV P2, #3; в порт P2 записано номер сторінки звертання MOV RO, #2; запис в покажчик адреси регістру зовнішньої пам'яті даних,

MOV A, #10; запис в акумулятор числа, що буде передаватися

MOVX @R0, A; передача числа з акумулятора на вказану адресу.

Якщо використовується двобайтовий покажчик (регістр DPTR), то можлива адресація до будь якого регістру зовнішньої пам'яті даних ємності 64К. Програма звертання в цьому випадку має наступний вигляд:

```
MOV DPH, #0A9H;     запис в регістр покажчик DPTR адреси комірки
MOV DPL, #04H;      пам'яті #0A904H зовнішньої пам'яті даних.
MOVX A,@DPTR;       передача байта даних в акумулятор.
```

Індексна адресація, або непряма регістрова по сумі базового та індексного регістрів, використовується для читання даних з пам'яті програм. Дозволяє використовувати табличні дані, що записані в пам'яті програм. Адреса будь якої комірки пам'яті може бути встановлена за сумою базової адреси, що задається двобайтовими регістрами покажчиками DPTR або PC, та адреси зміщення, що задається акумулятором ACC:

```
MOVC A, @A+PC; передача в акумулятор з адреси @A+PC
MOVC A, @ A+DPTR; передача в акумулятор з адреси @ A+ DPTR.
```

Символічна адресація широко використовується під час написання програм для найменування як регістрів так і окремих біт SFR регістрів користувача. Переважна більшість крос-асемблерних програм допускає вживання зарезервованих найменувань регістрів і біт SFR. Для встановлення оригінальних імен користувачем, можливе використання директив асемблера:

```
TIME_1DATA34H; регістру RAM з номером 34H присвоєне ім'я TIME1.
ON BIT 20H. 1; першому біту регістру 20H присвоєне ім'я ON.
```

В цьому випадку команди передачі даних мають наступний вигляд:

```
MOV TIME_1, A ; передача даних з акумулятора в регістр 34H.
SETB ON;  установка першого біту регістра 20H в "1".
```

Використання символічної адресації значно спрощує розробку програм та їх узгодження з мікроконтролерними системами на етапі конструювання.

Система команд Таблиця 4

Назва команди	Мнемокод	Б	Ц	Операція
Команди пересилання				
Пересилання в акумулятор з регістра (n = 0...7)	MOV A, Rn	1	1	(A) ← (Rn)
Пересилання в акумулятор прямоадресуємого байту	MOV A, ad	2	1	(A) ← (ad)
Пересилання в акумулятор байта з РПД (i = 0,1)	MOV A, @Ri	1	1	(A) ← РПД(Ri)
Завантаження в акумулятор константи	MOV A, #D8	2	1	(A) ← D8
Пересилання в регістр (n = 0...7) з акумулятора	MOV Rn, A	1	1	(Rn) ← (A)
Пересилання в регістр прямоадресуємого байта	MOV Rn, ad	2	2	(Rn) ← (ad)
Завантаження в регістр (n = 0..7) константи	MOV Rn, #D8	2	1	(Rn) ← D8
Пересилання по прямій адресі акумулятора	MOV ad, A	2	1	(ad) ← (A)
Пересилання по прямій адресі регістра	MOV ad, Rn	2	2	(ad) ← (Rn)
Пересилання прямоадресуємого байта за прямою	MOV add, ads	3	2	(add) ← (ads)
Пересилання байта з РПД по прямій адресі	MOV ad, @Ri	2	2	(ad) ← РПД(Ri)
Пересилання по прямій адресі константи	MOV ad, #D8	3	2	(ad) ← D8
Пересилання в РПД з акумулятора	MOV @Ri, A	1	1	РПД(Ri) ← (A)
Пересилання в РПД прямоадресуємого байта	MOV @Ri, ad	2	2	РПД (Ri) ← (ad)
Пересилання в РПД константи	MOV @Ri, #D8	2	1	РПД(Ri) ← D8
Завантаження регістра-покажчика даних	MOV DPTR,#D16	3	2	(DPTR) ← D16
Пересилання в акумулятор байта з ПП	MOVC A, @A+DPTR	1	2	(A) ← ПП (A + DPTR)
Пересилання в акумулятор байта з ПП	MOVC A, @A+PC	1	2	(PC) ← (PC) + 1 (A) ← ПП (A + PC)

Продовж. табл. 4

Пересилання в акумулятор байта з ЗПД	MOVX A, @Ri	1	2	(A) ← ЗПД(Ri)
Пересилання в акумулятор байта з розширеної ЗПД	MOVX A, @DPTR	1	2	(A) ← ЗПД(DPTR)
Пересилання у ЗПД з акумулятора	MOVX @Ri, A	1	2	ЗПД(Ri) ← (A)
Пересилання в розширену ЗПД з акумулятора	MOVX @DPTR, A	1	2	ЗПД(DPTR) ← (A)
Завантаження в стек	PUSH ad	2	2	(SP) ← (SP) + 1 РПД(SP) ← (ad)
Витяг зі стеку	POP ad	2	2	(ad) ← РПД(SP) (SP) ← (SP) - 1
Обмін акумулятора з регістром	XCH A, Rn	1	1	(A) ↔ (Rn)
Обмін акумулятора з прямоадресуємий байтом	XCH A, ad	2	1	(A) ↔ (ad)
Обмін акумулятора з байтом з РПД	XCH A, @Ri	1	1	(A) ↔ РПД(Ri)
Обмін молодшої тетради акумулятора з молодшою тетрадою байта РПД	XCHD A, @Ri	1	1	$(A_{0-3}) \leftrightarrow РПД(Ri)_{0-3}$
Арифметичні команди				
Додавання акумулятора з регістром (n = 0...7)	ADD A, Rn	1	1	(A) ← (A) + (Rn)
Додавання акумулятора з прямоадресуємим байтом	ADD A, ad	2	1	(A) ← (A) + (ad)
Додавання акумулятора з байтом із РПД (i = 0,1)	ADD A, @Ri	1	1	(A) ← (A) + РПД(Ri)
Додавання акумулятора з константою	ADD A, #D8	2	1	(A) ← (A) + D8
Додавання акумулятора з регістром і переносом	ADDC A, Rn	1	1	(A) ← (A) + (Rn) + (C)
Додавання акумулятора з прямоадресуємим байтом і	ADDC A, ad	2	1	(A) ← (A) + (ad) + (C)
Додавання акумулятора з байтом із РПД і переносом	ADDC A, @Ri	1	1	(A) ← (A) + РПД(Ri) +(C)

Продовж. табл. 4

Додавання акумулятора з константою і переносом	ADDC A, #D8	2	1	(A) ← (A) + D8 + (C)
Десяткова корекція акумулятора (після додавання операндів, представлених в упакованому BCD коді)	DA A	1	1	Якщо (A_{0-3}) > 9 V (AC) = 1, то (A_{7-0}) ← (A_{7-0}) + 6; потім, якщо ($A_{7.4}$) > 9 V (C)=1, то (A_{7-4})←(A_{7-4})+ 6
Віднімання від акумулятора регістра і займу	SUBB A, Rn	1	1	(A) ← (A) - (C) - (Rn)
Віднімання від акумулятора прямоадресуємого байта і займу	SUBB A, ad	2	1	(A) ← (A) - (C) - (ad)
Віднімання з акумулятора байта РПД і займу	SUBB A, @Ri	1	1	(A) ← (A) - (C) - РПД(Ri)
Віднімання з акумулятора константи і займу	SUBB A, #D8	2	1	(A) ← (A) - (C) - D8
Інкремент акумулятора	INC A	1	1	(A) ← (A) + 1
Інкремент регістра	INC Rn	1	1	(Rn) ← (Rn) + 1
Інкремент прямоадресуємого байта	INC ad	2	1	(ad) ← (ad) + 1
Інкремент байта в РПД	INC @Ri	1	1	РПД(Ri) ← РПД(Ri) + 1
Інкремент покажчика даних	INC DPTR	1	2	(DPTR) ← (DPTR) + 1
Декремент акумулятора	DEC A	1	1	(A) ← (A) - 1
Декремент регістра	DEC Rn	1	1	(Rn) ← (Rn) - 1
Декремент прямоадресуємого байта	DEC ad	2	1	(ad) ← (ad) - 1
Декремент байта в РПД	DEC @Ri	1	1	РПД(Ri) ← РПД(Ri) - 1
Множення акумулятора на регістр В	MUL AB	1	4	(B)(A) ← (A) x (B)
Ділення акумулятора на регістр В	DIVAB	1	4	(A)←(A)/(B) - ціла частина, (B)← R((A)/(B)) залишок від ділення
Логічни команди				
Логічне І акумулятора і регістра	ANL A, Rn	1	l	(A) ← (A) Λ (Rn)
Логічне І акумулятора і прямоадресуємого байта	ANL A, ad	2	l	(A) ← (A) Λ (ad)

Продовж. табл. 4

Логічне І акумулятора і байта з РПД	ANL A, @Ri	1	l	(A) ← (A) Λ РПД(Ri)
Логічне І акумулятора і константи	ANL A, #D8	2	l	(A) ← (A) Λ D8
Логічне І прямоадресуємого байта й акумулятора	ANL ad, A	2	1	(ad) ← (ad) Λ (A)
Логічне І прямоадресуємого байта і константи	ANL ad, #D8	3	2	(ad) ← (ad) Λ D8
Логічне АБО акумулятора і регістра	ORL A,Rn	1	1	(A) ← (A) V (Rn)
Логічне АБО акумулятора і прямоадресуємого байта	ORL A, ad	2	1	(A) ← (A) V (ad)
Логічне АБО акумулятора і байта з РПД	ORL A, @Ri	1	1	(A) ← (A) V РПД(Ri)
Логічне АБО акумулятора і константи	ORL A, #D8	2	1	(A) ← (A) V D8
Логічне АБО прямоадресуємого байта й акумулятора	ORL ad, A	2	1	(ad) ← (ad) V (A)
Логічне АБО прямоадресуємого байта і константи	ORL ad, #D8	3	2	(ad) ← (ad) V D8
Виключальне АБО акумулятора і регістра	XRL A,Rn	1	1	(A) ← (A) ⊕ (Rn)
Виключальне АБО акумулятора і прямоадресуємого байта	XRL A, ad	2	1	(A) ← (A) ⊕ (ad)
Виключальне АБО акумулятора і байта РПД	XRL A, @Ri	1	1	(A) ← (A) ⊕ РПД(Ri)
Виключальне АБО акумулятора і константи	XRL A, #D8	2	1	(A) ← (A) ⊕ D8
Виключальне АБО прямоадресуємого байта та акумулятора	XRL ad, A	2	1	(ad) ← (ad) ⊕ (A)
Виключальне АБО прямоадресуємого байта і константи	XRL ad, #D8	3	2	(ad) ← (ad) ⊕ D8
Скидання акумулятора	CLR A	1	1	(A) ← 0
Інверсія акумулятора	CPL A	1	1	(A) ← ($\overline{A}$)
Обмін тетрад в акумуляторі	SWAP A	1	1	(A_{3-0}) ↔ (A_{7-4})

Продовж. табл. 4

Зсув акумулятора ліворуч циклічний	RL A	1	1	$(A_{n+1}) \leftarrow (A_n)$, n = 0...6, $(A_o) \leftarrow (A_7)$
Зсув акумулятора ліворуч через перенос	RLC A	1	1	$(A_{n+1}) \leftarrow (A_n)$, n = 0...6, $(A_o) \leftarrow (C)$, $(C) \leftarrow (A_7)$
Зсув акумулятора праворуч циклічний	RR A	1	1	$(A_n) \leftarrow (A_{n+1})$, n = 0...6, $(A_7) \leftarrow (A_o)$
Зсув акумулятора праворуч через перенос	RRC A	1	1	$(A_n) \leftarrow (A_{n+1})$, n = 0...6, $(A_7) \leftarrow (C)$, $(C) \leftarrow (A_o)$
Команди операцій з бітами				
Скидання переносу	CLR C	1	1	(C) ← 0
Скидання біта	CLR bit	2	1	(bit) ← 0
Встановлення переносу	SETB C	1	1	(C) ← 1
Установка біта	SETB bit	2	1	(bit) ← 1
Інверсія переносу	CPL C	1	1	$(C) \leftarrow (\overline{C})$
Інверсія біта	CPL bit	2	1	$(bit) \leftarrow (\overline{bit})$
Логічне І переносу і біта	ANL C, bit	2	2	$(C) \leftarrow (C) \wedge (bit)$
Логічне І переносу та інверсії біта	ANL C, /bit	2	2	$(C) \leftarrow (C) \wedge (\overline{bit})$
Логічне АБО переносу і біта	ORL C, bit	2	2	$(C) \leftarrow (C) \vee (bit)$
Логічне АБО переносу та інверсії біта	ORL C, /bit	2	2	$(C) \leftarrow (C) \vee (\overline{bit})$
Пересилання біта у перенос	MOV C, bit	2	1	(C) ← (bit)
Пересилання переносу у біт	MOV bit, C	2	2	(bit) ← (C)
Команди передавання керування				
Довгий перехід у межах всієї пам'яті програм	LJMP ad16	3	2	(PC) ← ad16
Абсолютний перехід в межах сторінки 2 Кбайта	AJMP ad11	2	2	$(PC_{0\text{-}10}) \leftarrow$ ad11
Короткий відносний перехід в межах сторінки 256 б	SJMP ad8	2	2	(PC) ← (PC) + 2 (PC) ← (PC) + ad8
Непрямий перехід	JMP @A + DPTR	1	2	(PC) ← (A) + (DPTR)
Перехід, якщо акумулятор дорівнює нулю	JZ ad8	2	2	(PC) ← (PC+2), якщо (A)=0, то (PC)←(PC)+ad8
Перехід, якщо акумулятор не дорівнює нулю	JNZ ad8	2	2	(PC) ← (PC)+2, якщо (A)≠ 0, то (PC) ← (PC) + ad8

Продовж. табл. 4

Перехід, якщо перенос дорівнює одиниці	JC ad8	2	2	(PC) ← (PC)+2, якщо (C)=1, то (PC)←(PC)+ad8
Перехід, якщо перенос дорівнює нулю	JNC ad8	2	2	(PC) ← (PC)+2, якщо (C) = 0, то (PC) ← (PC) + ad8
Перехід, якщо біт дорівнює одиниці	JB bit, ad8	3	2	(PC) ← (PC)+3, якщо (bit) = 1, то (PC) ← (PC) + ad8
Перехід, якщо біт дорівнює нулю	JNB bit, ad8	3	2	(PC)← (PC)+3, якщо (bit) = 0, то (PC) ← (PC) + ad8
Перехід, якщо біт установлений, з наступним скиданням біта	JBC bit,ad8	3	2	(PC) ← (PC)+3, якщо (bit)= 1, то (bit)← 0 і (PC) ← (PC) + ad8
Декремент регістра і перехід, якщо не нуль	DJNZ Rn, ad8	2	2	(PC) ← (PC)+2, (Rn) ← (Rn) - 1, якщо (Rn) ≠ 0, то (PC) ← (PC) + ad8
Декремент прямоадресуємого байта і перехід, якщо не нуль	DJNZ ad,ad8	3	2	(PC) ← (PC) + 3, (ad) ← (ad) - 1, якщо (ad) ≠ 0, то (PC) ← (PC) + ad8
Порівняння акумулятора з прямоадресуємим байтом і перехід, якщо не дорівнює	CJNE A, ad, ad8	3	2	(PC) ← (PC)+3, якщо (A)≠(ad), то (PC) ← (PC) + ad8, якщо (A)<(ad), то (C) ← 1, інакше (C) ← 0
Порівняння акумулятора з константою і перехід, якщо не дорівнює	CJNE A, #D8, ad8	3	2	(PC)←(PC)+3, якщо (A)≠ D8, то (PC)←(PC)+ad8, (A)<D8, то (C)←1, інакше (C) ← 0
Порівняння регістра з константою і перехід, якщо не дорівнює	CJNE Rn, #D8, ad8	3	2	(PC)←(PC)+3, якщо (Rn) ≠ D8, то (PC)←(PC)+ ad8, якщо (Rn) < D8, то (C) ← 1, інакше (C) ← 0
Порівняння байта в РПД із константою і перехід, якщо не дорівнює	CJNE @Ri, #D8, ad8	3	2	(PC) ← (PC) + 3, якщо РПД(Ri) ≠ D8, то (PC) ← (PC) + ad8, якщо РПД(Ri) < D8, то (C) ←1, інакше (C) ← 0
Довгий виклик підпрограми	LCALL ad16	3	2	(PC)←(PC)+3, (SP)← (SP)+l, РПД(SP)←($PC_{7\text{-}0}$), (SP)←(SP)+1, РПД(SP) ← ($PC_{8\text{-}15}$), (PC) ← ad16

Продовж. табл. 4

Абсолютний виклик підпрограми в межах сторінки в 2 Кбайта	ACALL ad11	2	2	(PC) ← (PC)+2, (SP) ← (SP)+1, РПД(SP) ← ($PC_{7\text{-}0}$), (SP) ← (SP)+1, РПД(SP) ← ($PC_{8\text{-}15}$), (PC) ← ad11
Повернення з підпрограми	RET	1	2	($PC_{15\text{-}8}$) ← РПД(SP), (SP) ←(SP)-1, ($PC_{7\text{-}0}$)← РПД(SP), (SP)← (SP) – 1
Повернення з підпрограми обробки переривання	RETI	1	2	($PC_{15\text{-}8}$) ← РПД(SP), (SP) ← (SP)-1, ($PC_{7\text{-}0}$) ← РПД(SP), (SP)←(SP) - 1
Порожня команда	NOP	1	1	(PC) ← (PC) + 1

Вплив команд на прапорці ілюструє табл. 5, у якій позначено: «+» — команда впливає на прапорець, «-» — не впливає; «1» — встановлює у стан «1»; «0» — скидає у стан «0».

Вплив команд на прапорці Таблиця 5

Операція	Команда	Прапорці		
		OV	C	AC
Пересилання	MOV PSW, source	+	+	+
Додавання	ADD, ADDC, SUBB	+	+	+
Множення	MUL	+	0	-
Ділення	DIV	+	0	-
Порівняння і перехід	CJNE	-	+	-
Десяткова корекція	DAA	-	+	+
Зсув	RRC, RLC	-	+	-
Керування прапорцями	SETB C	-	1	-
	CLR C	-	0	-
	CPL C	-	$\overline{C}$	-
Команди роботи з бітами	ANL C, bit	-	+	-
	ANL C,/bit	-	+	-
	ORL C, bit	-	+	-
	ORL C,/bit	-	+	-
	MOV C, bit	-	+	-

Примітка. Прапорець парності P встановлюється за кожною командою, що модифікує вміст акумулятора згідно із цим вмістом. Прапорець нуля Z фізично не існує, однак під час виконання команд JZ, JNZ відбувається порівняння A з нулем.

ЛАБОРАТОРНА РОБОТА № 2

Виведення аналогових сигналів у мікроконтролерах сімейства MCS-51.

1. Мета роботи:

Ознайомитися з методами виведення аналогових сигналів у мікроконтролерах сімейства MCS-51 з зовнішніми цифро-аналоговими перетворювачами.

2. Програма роботи:

2.14.Ознайомитися з платою розширення лабораторного стенда “EV8031/AVR”.

2.15.Ознайомитися з системою команд мікроконтролерів сімейства MCS-51.

2.16.Скласти програму на мові Асемблер для виведення аналогового періодичного сигналу заданої форми та параметрів з періодом T (рис. 1). Аналоговий сигнал формувати на виході 8-розрядного цифро-аналогового перетворювача лабораторного стенда (адреса цифро-аналогового перетворювача F000H).

2.17.На персональному комп'ютері завантажити текстовий редактор (Total Commander).

2.18.У текстовому редакторі набрати текст програми в мнемокодах мови Ассемблер для MCS-51.

2.19.Зберегти набраний файл із розширенням *.ASM.

2.20.Відкомпілювати набрану програму відповідними засобами (наприклад, компілятором ASM51).Для компіляції у командному рядку Total Commander набрати: ASM51.EXE NAME.ASM, де NAME - ім’я збереженого файлу (файли ASM51 та NAME повинні знаходитися в одній директорії).

2.21.Можливі помилки в програмі можна переглянути в однойменному файлі з розширенням *.LST.

2.22.Після усунення всіх помилок, дані файлу з розширенням *.HEX програмою EVAL32.EXE необхідно перенести в стенд. Для цього у командному

рядку Total Commander набрати: EVAL32.EXE –hs –com 1 9600 NAME.HEX.

2.23. Вивід на екран підказки про параметри програми EVAL32.EXE, здійснюється запуском EVAL32.EXE.

2.24.При передаванні даних з персонального комп'ютера в лабораторний стенд на екрані монітора відображаються дані, що передаються. Ці ж дані відображаються на індикаторі стенда HL1. Горить світлодіод HL9. Після передавання останнього байта завантажена програма запускається автоматично.

2.25.При необхідності перезапуску завантаженої в стенд програми натиснути кнопку SW1.

2.26.Зупинка завантаженої програми та перехід у режим очікування на прийом даних з персонального комп'ютера можливо натисканням кнопки SW2. При цьому гасне світлодіод HL9. Запис нової програми можливий в будь-який момент часу роботи завантаженої програми.

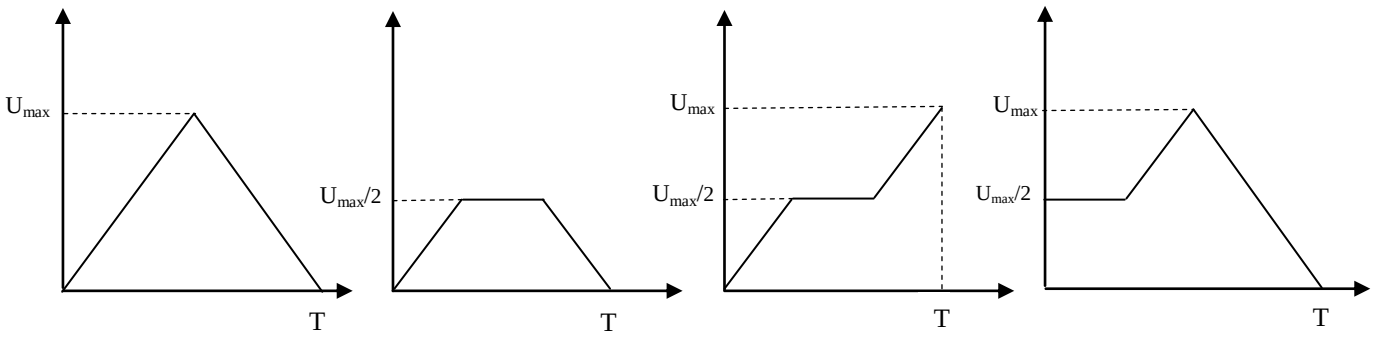


Рис. 1. Форма та параметри аналогових сигналів

4. Зміст звіту

- Титульний лист з відомостями про назву роботи і склад бригади.
- Текст програми з коментарями.

5. Контрольні запитання

- Як реалізується виведення аналогових сигналів?
- Які існують методи цифро-аналогового перетворювання та різновиди

цифро-аналогових перетворювачів?

- Назвіть параметри цифро-аналогових перетворювачів.

6. Опис плати розширення лабораторного стенда.

Плата розширення у складі лабораторного стенда призначена для проведення лабораторних робот, пов'язаних з цифро-аналоговим, аналого-цифровим та частотним перетворенням, а також з обробкою дискретних сигналів. Структурна схема плати розширення наведена на рис. 2.

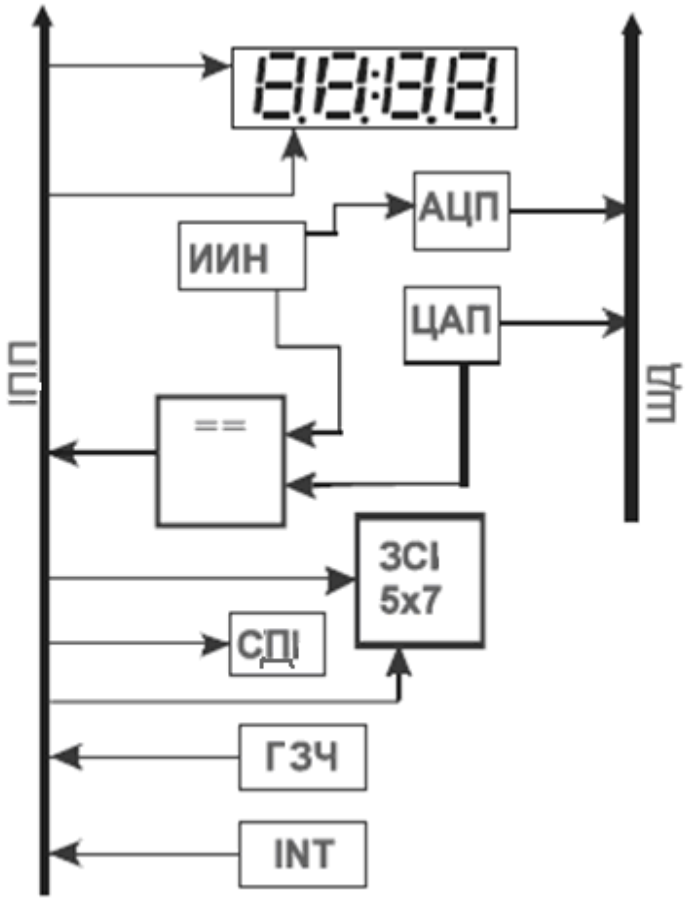


Рис. 2. Структурна схема плати розширення:

8888 - 4 розрядна динамічна індикація;

ІПП- інтерфейс периферійних пристроїв;

ЦАП – цифро-аналоговий перетворювач;

СДІ - світлодіодні індикатори;

ЗСІ - знакосинтезуючий індикатор 5х7;

ГЗЧ - генератор зі змінюваною частотою генерації;

INT - кнопки запиту переривання;

ДВН - джерело вимірюваної напруги;

ШД - шина даних.

Цифро-аналоговий перетворювач.

Цифро-аналоговий перетворювач (ЦАП) виконаний на мікросхемі AD7801 (8-розрядний ЦАП). Вхідними сигналами для ЦАП є лінії AD7-AD0. Вихідний сигнал знімається з рознімання BNC. Адреса звернення – F000H.

Аналого-цифровий перетворювач.

АЦП виконаний на мікросхемі ЦАП AD7801, операційному підсилювачі, використовуваним як компаратор LM358 DA1. Вхідним аналоговим сигналом для АЦП є сигнал зі змінного резистора R19. Лінії AD7-AD0 (див. схему стенда) використовуються для формування цифрового вхідного коду. На виході ЦАП формується напруга, пропорційне вхідному коду. При спрацьовуванні компаратора загоряється світлодіод HL1. У розширеній комплектації стенда поставляється мікросхема AD7813 - 8-розрядний АЦП.

Генератори.

У схемі присутній генератор зі змінюваною частотою генерації ~ 1- 50 кгц, елементи R1, R4, R5, R7, R10, R11, R15, R16, C3, VT1, DA1-1 (зміна частоти здійснюється за допомогою резистора R4), і генератор з фіксованою частотою ~ 10кгц, елементи R19, R20, C16, DD18-1, DD18-2, DD18-3.

Вивід дискретної інформації.

Вивід дискретної інформації здійснюється за допомогою чотирьохрозрядного семисегментного індикатора HL2, який включений за схемою динамічної індикації.

Схема розташування елементів плати розширення наведена на рис. 3:

HG1 - знакосинтезуючий індикатор 5x7;

HL2 - 4-х розрядна динамічна індикація;

HL1 - світлодіодний індикатор спрацьовування компаратора;

J1 - перемичка підключення до рознімання J2 виходи генератора з постійною частотою генерації;

J2 - рознімання підключення зовнішніх контрольно-вимірювальних приладів;

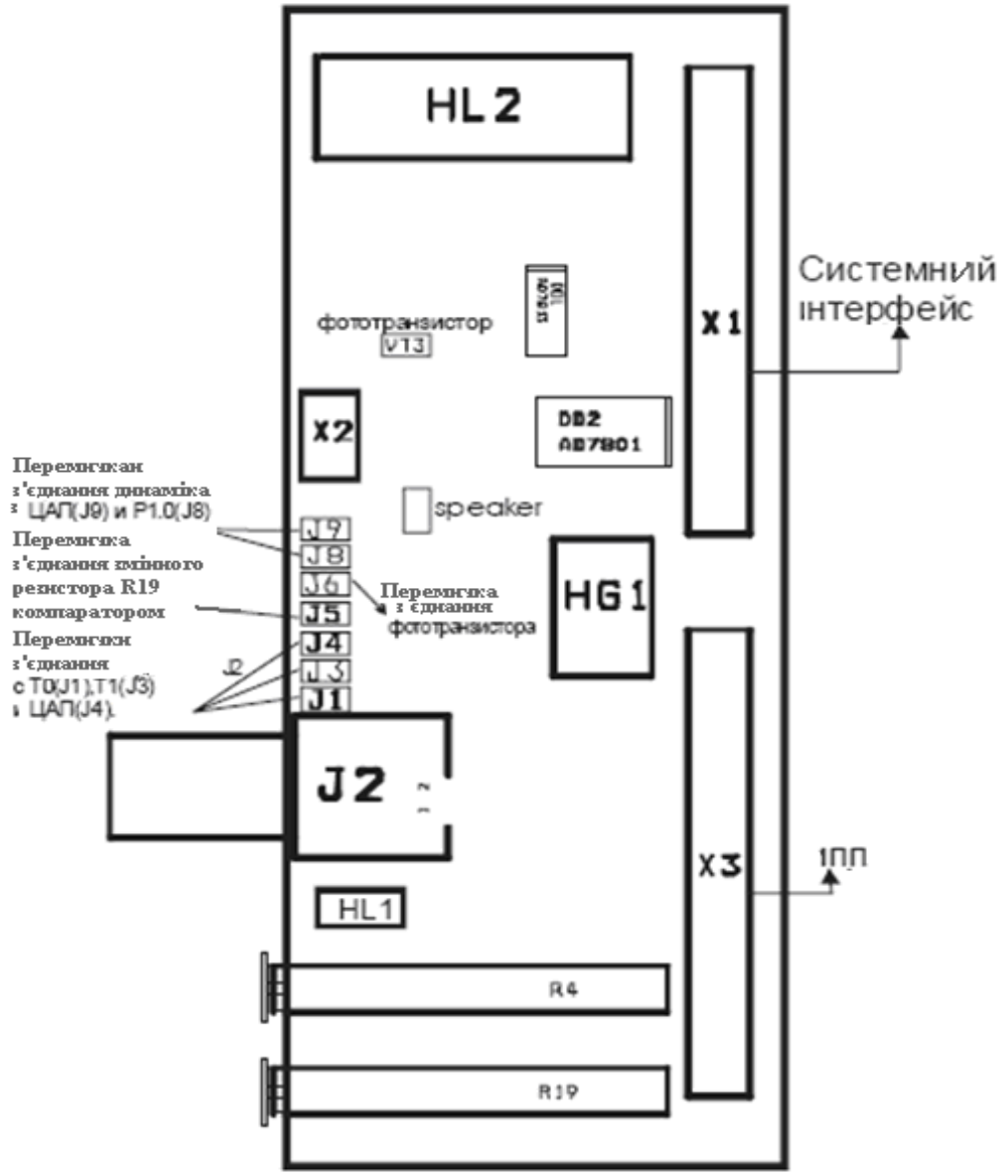


Рис. 3. Схема розташування елементів плати розширення

J3 - перемичка підключення до рознімання J2 виходи генератора зі змінною частотою генерації;

J4 - перемичка підключення до рознімання виходу ЦАП;

J5 - підключення як джерело зовнішнього переривання INT1 кнопки S11;

J6 - підключення як джерело зовнішнього переривання INT1 зовнішнього джерела який може бути підключений через рознімання JP1;

J7 - інтерфейс підключення плати розширення до стенда;

J8 - підключення до входу АЦП зовнішнього джерела сигналу, підключеного до розніманню JP2;

J9 - підключення як джерело сигналу для АЦП змінного резистора R27;

R19 - змінний резистор, джерело вхідного сигналу для АЦП;

R4 - змінний резистор, змінює частоту генерації генератора імпульсів.

7. Система команд мікроконтролерів сімейства MCS-51

Система команд МК i8051 містить 111 команд, які поділяють на п'ять груп: команди пересилання; арифметичні команди; логічні команди; команди передавання керування; команди операцій з бітами. Команди оперують з одно-, чотири-, вісьми- та 16-бітними словами.

Більшість команд (94) мають формат 1 - 2 байт і виконуються за один-два цикли (за тактової частоти 12 МГц тривалість циклу дорівнює 1 мкс). Систему команд МК i8051 наведено у інструкції до лабораторної роботи № 1.

ЛАБОРАТОРНА РОБОТА № 3

Інтегроване середовище розробки AVR Studio. Написання та відладка простих програм для AVR-мікроконтролерів.

Мета роботи – ознайомитись з послідовністю створення та відладки програмного забезпечення мікроконтролерів сімейства AVR фірми Atmel у інтегрованому середовищі розробки AVR Studio.

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

AVR Studio 4 є інтегрованим середовищем розробки для написання та відладки програмного забезпечення мікроконтролерів сімейства AVR. AVR Studio 4 надає засоби управління проектами, редагування та асемблювання вихідних кодів. Також AVR Studio 4 дає змогу проводити симуляцію роботи програми, програмування кристалу мікроконтролера, здійснювати апаратну відладку.

1.1. Робота в інтегрованому середовищі розробки AVR Studio 4

1.1.1. Створення проекту та ввкдення програми

Запустіть AVR Studio. Виберіть розділ меню **Project/New Project**. Вікно, яке з'явиться при цьому **(**рис. 1) дає змогу створити новий проект. Присвоїмо ім'я 1 назві проекту (поле **Project Name**) та асемблерному файлу (поле **Initial File**). Оскільки прапорець **Create Folder** встановлений, то при натисненні кнопки **Next>>** AVR Studio автоматично створить папку з назвою 1, асемблерним файлом 1.asm та файлом проекту 1.aps у каталозі, який задається полем **Location**. Бажано проекти зберігати у тому ж каталозі, де інстальована програма **AVR Studio**.

У полі **Project Type** виберіть тип проекту **Atmel AVR Assembler** і натисніть кнопку **Next>>**. У вікні **Select debug platform and device**, що при цьому появиться (**Рис. 2**), виберіть у полі **Debug Platform** рядок **AVR Simulator**, потім у полі **Device** рядок **ATmega8** – мікроконтролер, для якого створюється програма і натисніть кнопку **Finish**.

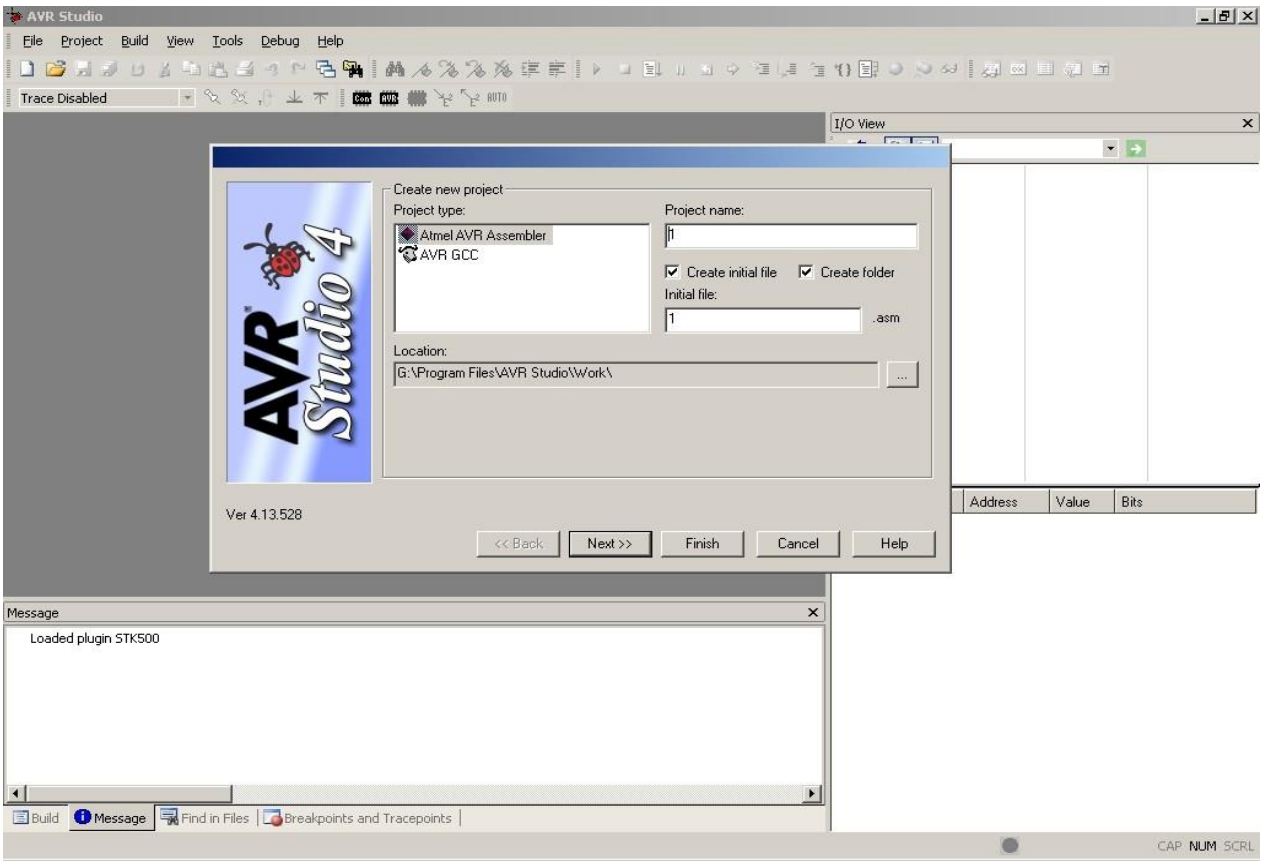


Рис. 1. Створення проекту в AVR Studio

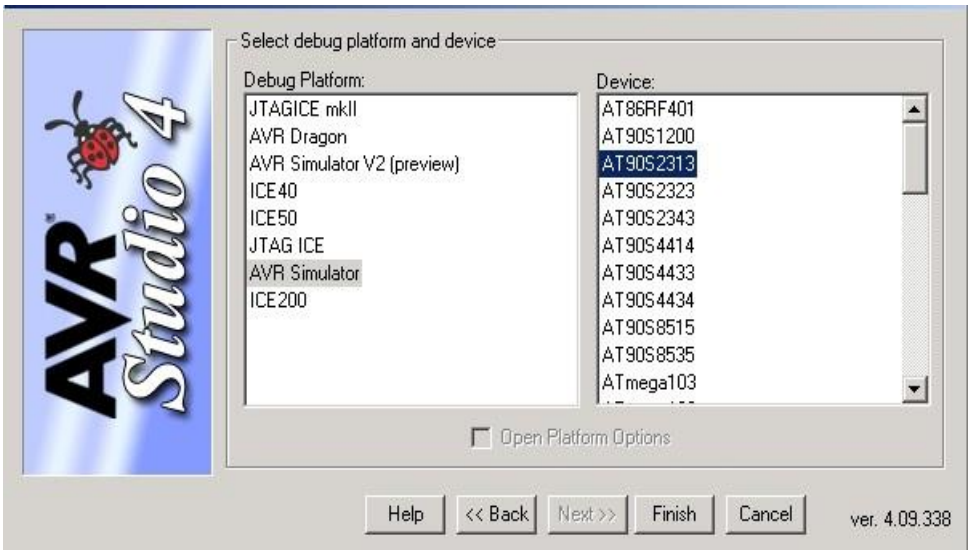


Рис. 2. Налаштування проекту в AVR Studio

Тепер на екрані буде активне вікно G:\Program Files\AVR Studio\Work\1.asm. У ньому набирається і редагується текст програми на мові асемблера.

Під час введення програми можна використовувати всі прийоми роботи з текстом, знайомі з досвіду роботи з Microsoft Word та іншими текстовими

редакторами, що підтримуються пунктами меню **Edit**.

Щоб переглянути всю систему команд МК **ATmega8**, або особливості виконання певної команди потрібно в **AVR Studio** вибрати пункт меню **Help/AVR Tools User Guide**. При цьому з'явиться вікно AVR Tools (**Рис. 3**), в якому потрібно спочатку вибрати пункт **AVR Assembler/Parts/ATmega8 Instruction Set**. Натиснувши мишкою на певну команду можна отримати детальну інформацію про виконання команди та типові приклади її використання.

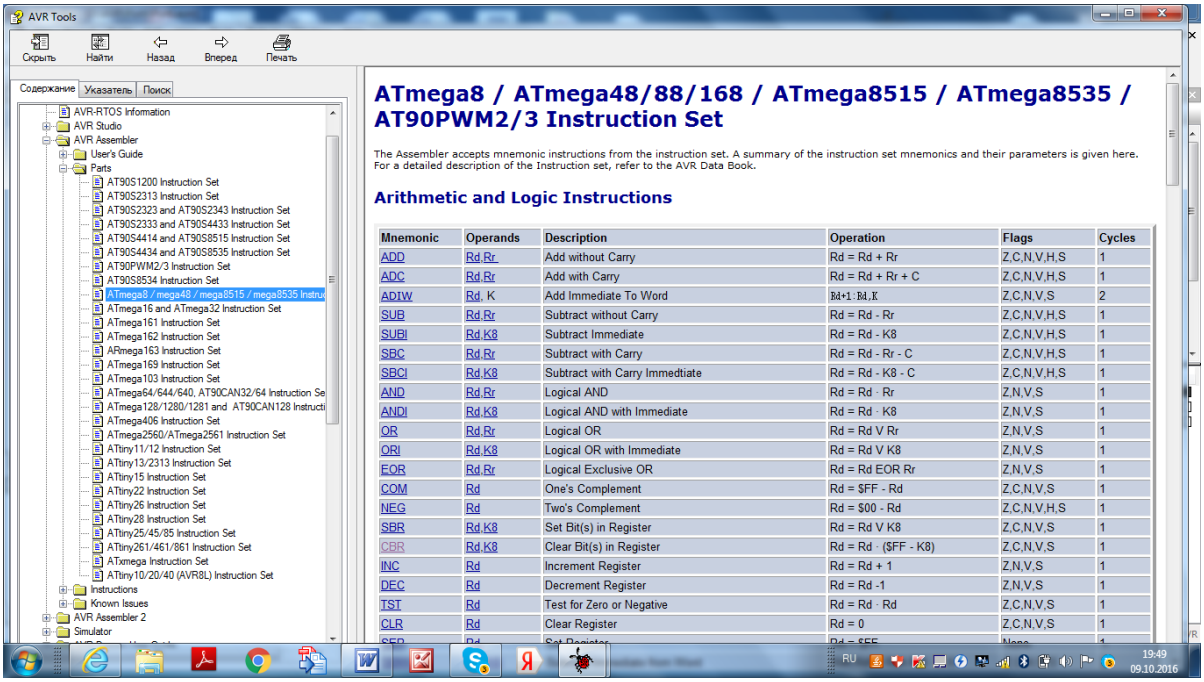


Рис. 3. Вікно довідки по системі команд МК ATmega8

1.1.2. Асемблювання програми

Після завершення вводу програми, перед її симуляцією або записом у пам'ять МК, необхідно провести її асемблювання. Результат асемблювання програми може виводитись у різних форматах. За замовчуванням результат зберігається у файл з розширенням *.hex, і саме цей формат файлу (Intel Intellec 8/MDS) може використовуватись програматорами длязавантаження програм у пам'ять МК. Формат файлу задається через пункт меню **Project/AVR Assembler Setup,** при цьому з'являється вікно (**Рис. 4**), де у полі **Additional Output file format** вибирається потрібне значення.

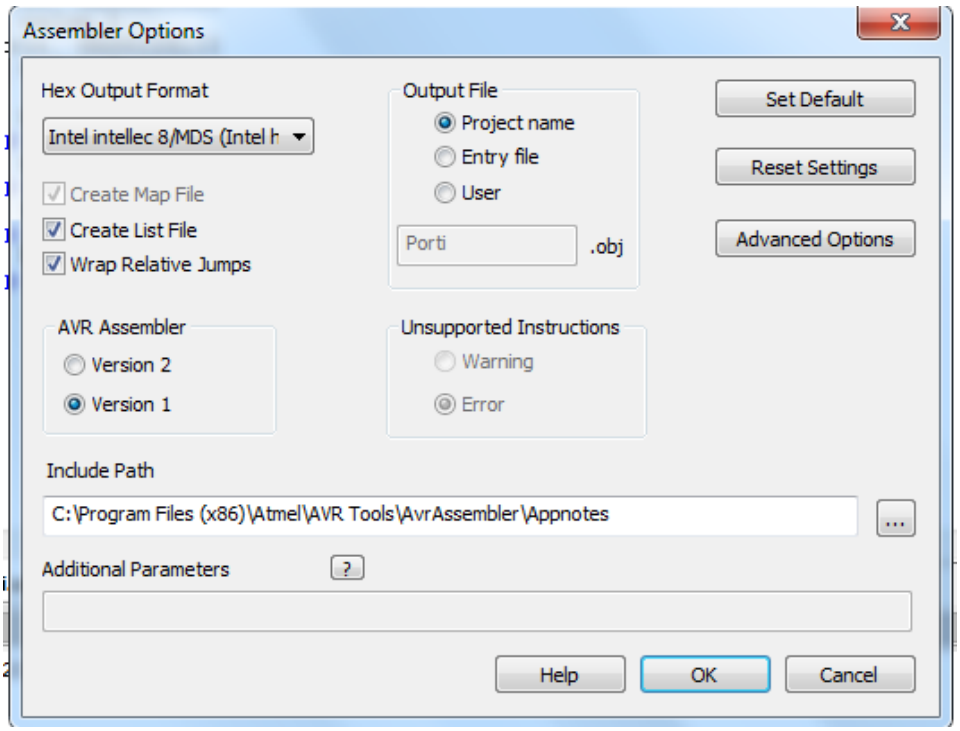


Рис. 4. Вікно управління асемблюванням

Для асемблювання програми можна скористатися пунктом меню **Project/Build**, клавішою F7 або кнопкою **Build** панелі інструментів. У нижньому вікні **Output** виводиться інформація про результат асемблювання (**Рис. 5**).

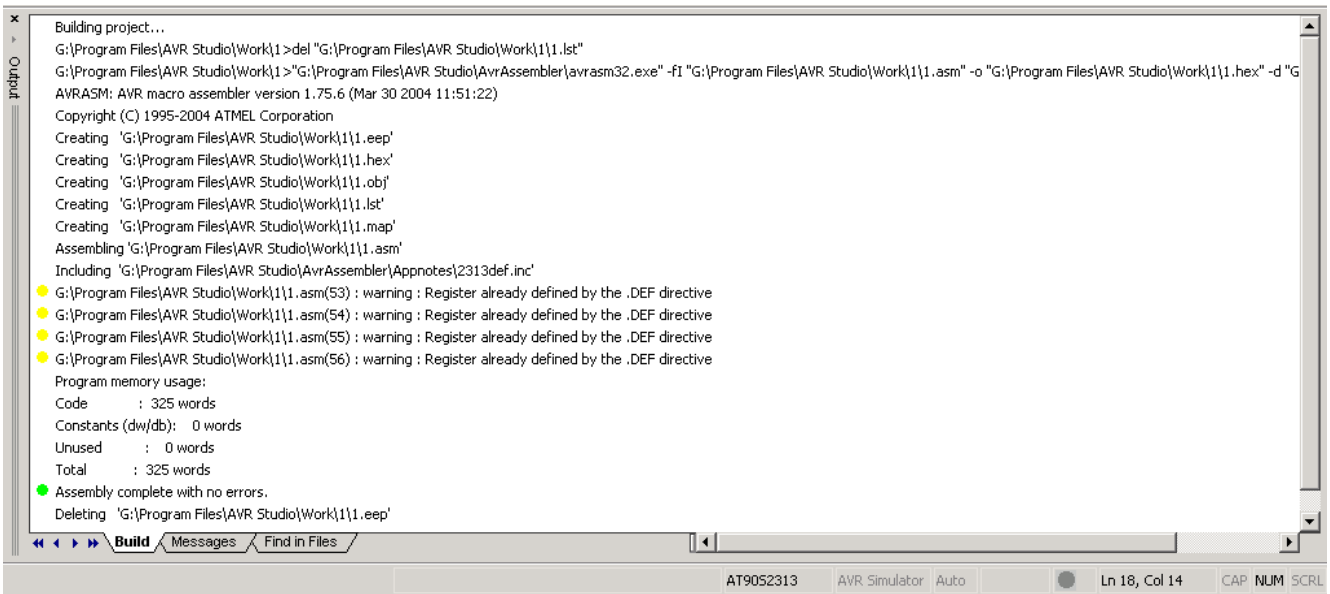


Рис. 5. Вікно Output з повідомленнями про результат асемблювання програми

У вікні виводиться така інформація:

Assembling 'G:\Program Files\AVR Studio\Work\1\1.asm' – ім’я файлу, що

асемблюється;

Including 'G:\Program Files\AVR Studio\AvrAssembler\Appnotes\2313def.inc' – файли, включені у програму директивою .include.

Code: 325 words – розмір програми у 16-розрядних словах;

Constants (dw/db): 0 words – кількість констант розміщених у пам'яті програм директивами .dw або .db;

Total: 325 words – загальний розмір використаної пам'яті програм;

Assembly complete with no errors – повідомлення про те, що помилки не виявлені.

Попередження виводяться у рядках позначених жовтими кружками, помилки – червоними.

Якщо в програмі виявлені помилки, вміст вікна **Output** дає змогу їх локалізувати та визначити причину помилки. Кожній помилці (error) відповідає повідомлення, представлене одним рядком у вікні **Output** (**Рис. 6**).

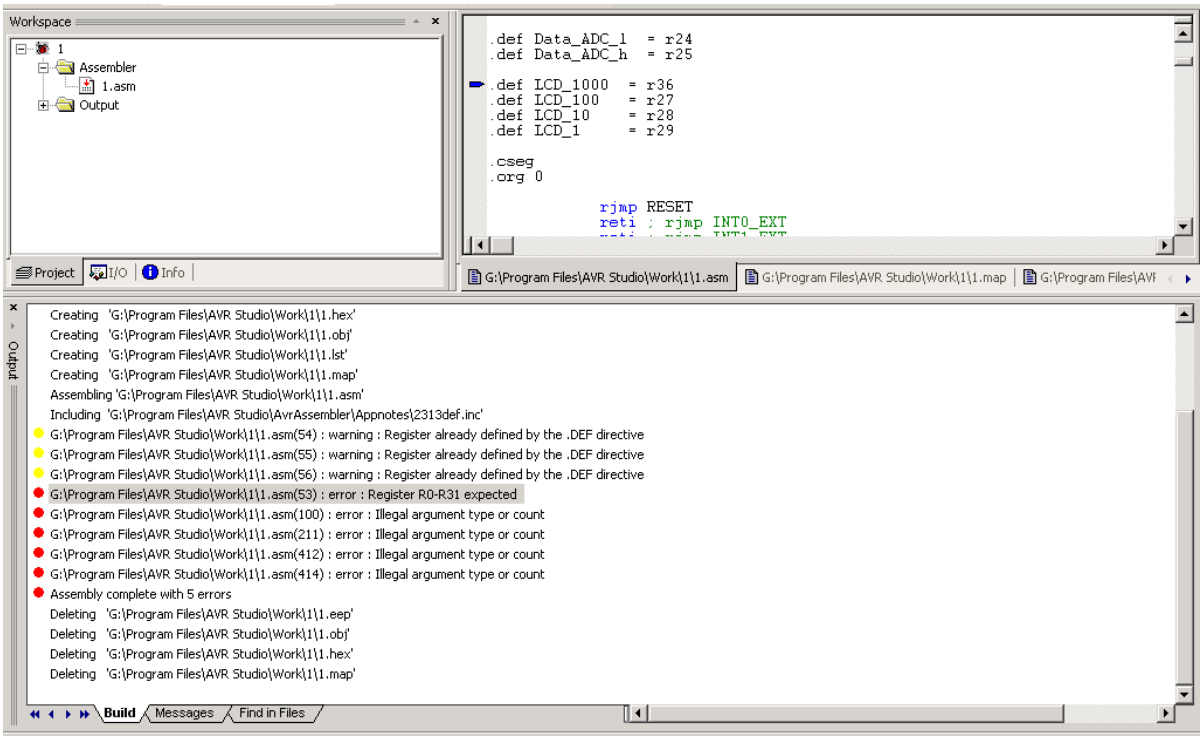


Рис. 6. Вид вікна Output з повідомленнями про помилки

Наприклад:

G:\Program Files\AVR Studio\Work\1\1.asm(53) : error : Register R0-R31 expected

Це означає, що у файлі G:\Program Files\AVR Studio\Work\1\1.asm у рядку 53 виявлена помилка: можуть бути використані регістри r0…r31, а є спроба використати неіснуючий регістр r36.

Якщо двічі клацнути мишкою по рядку з повідомленням про помилку у вікні **Output** то курсор автоматично встановиться на рядок програми з помилкою, а також напроти цього рядку у вікні 1.asm з'явиться синя стрілка. Номер рядку в якому розташовано курсор також можна визначити по значенню Ln 53, Col 1 (рядок 53, колонка 1), яке виводиться у нижньому правому куті вікна **AVR Studio** (**Рис. 5**).

У разі наявності повідомлень про помилки, необхідно усунути всі помилки і повторити асемблювання.

Після успішного асемблювання у вікні **Workspace** на вкладці **Project** появиться папка **Output**, в якій буде розміщено файли лістингу 1.lst і карти пам'яті 1.map. Файл лістингу містить поряд з рядками асемблерного тексту також адреси комірок пам'яті, в яких буде розташовано дану команду (перша колонка), а також її шістнадцятковий код (друга колонка) - **Рис. 7**. Дозволити або заборонити створення цих файлів можна через пункт меню **Project/ Assembler Options.** при цьому з'являється вікно (**Рис. 4**), де у полях **Map file** та **List file** потрібно встановити або зняти відповідні прапорці.

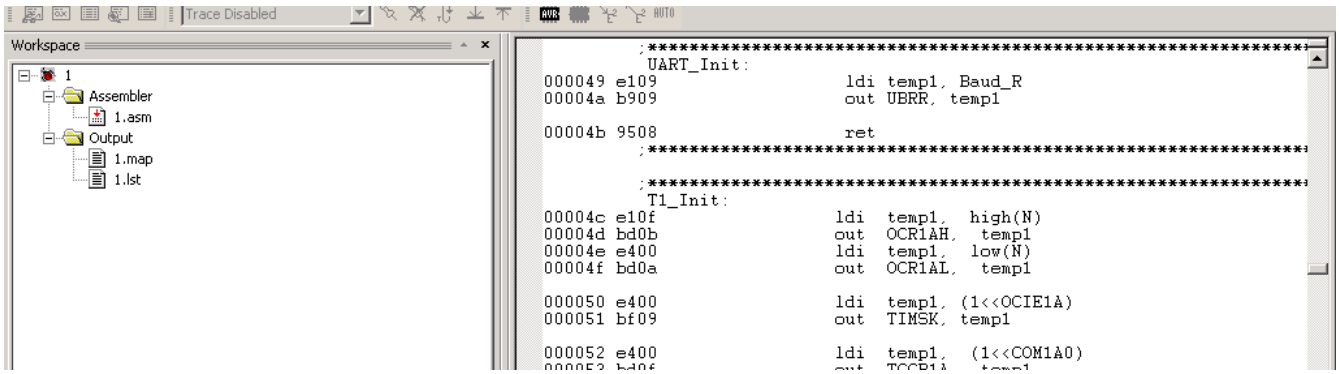


Рис. 7. Формат файлу лістингу *.lst

1.1.3. Відладка програми

Відладка програми полягає у її покроковому виконанні з контролем стану внутрішніх змінних МК, правильності конфігурації периферійних модулів,

послідовності виклику підпрограм і переривань та реакції на зовнішні події.

Після успішного асемблювання для запуску покрокової відладки виберіть пункт меню **Debug/Start Debugging**. У вікні програми, зліва від першої команди, з'явиться жовта стрілка, яка вказує поточну команду, що виконується.

Після цього можна задати опції симулятора **AVR Studio**. Для цього потрібно вибрати пункт меню **Debug/AVR Simulator Options** (**Рис. 8**).

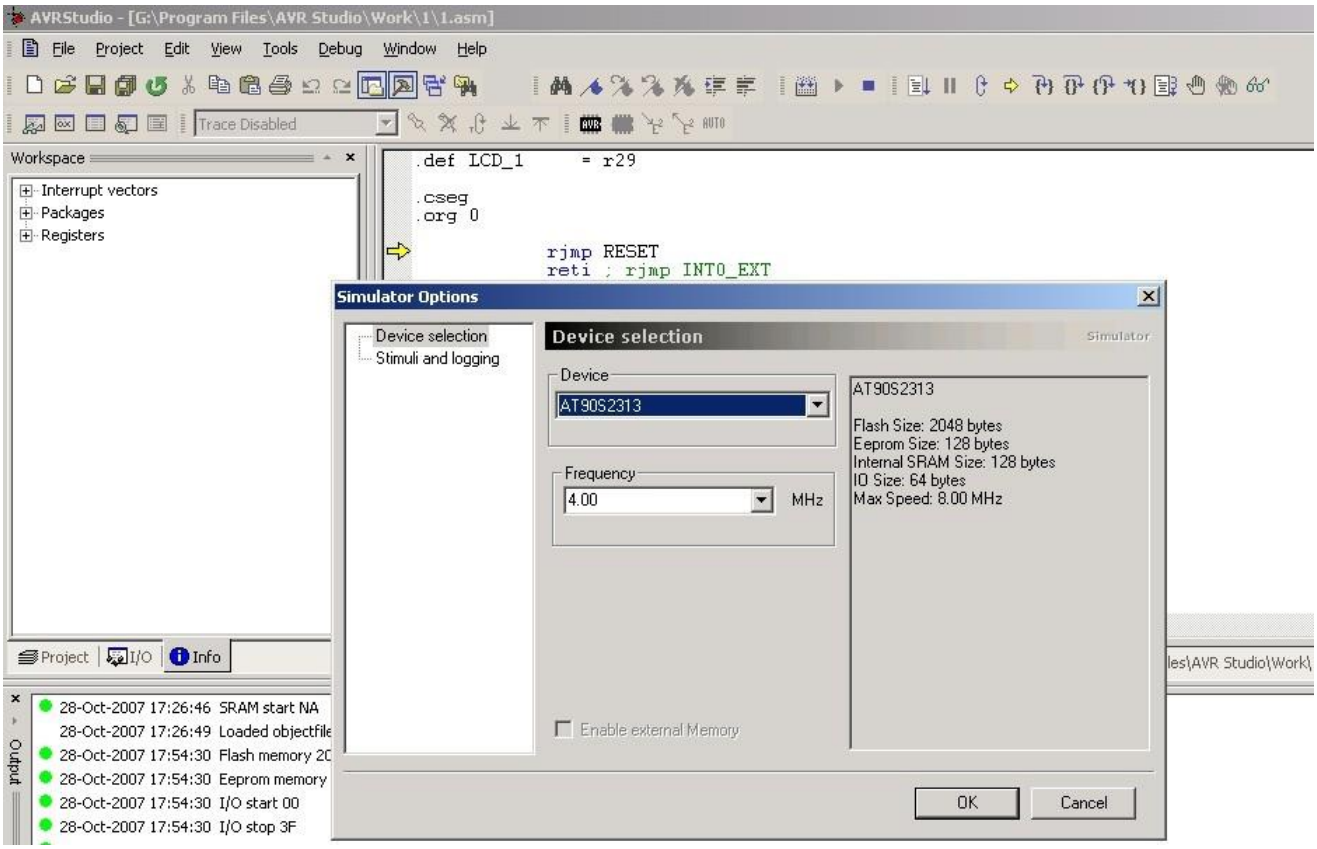


Рис. 8. Вікно налаштування параметрів симуляції

В полі **Frequency** вікна **Simulator Options** можна вказати робочу частоту МК. Для нашого випадку ця частота рівна 4 МГц і відповідає резонансній частоті кварцового резонатора підключеного до виводів XTAL1-XTAL2 МК.

У вікні **Processor** виводиться детальна інформація про стан ядра МК (**Рис. 9, а**), а саме: вибрана тактова частота МК (поле **Frequency**), значення лічильника кількості тактів (поле **Cycle Counter**), вміст індексних регістрових пар X (r26-r27), Y(r28-r29), Z (r30-r31), вказівника стеку (поле **Stack Pointer**), лічильника команд (поле **Program Counter**), регістру стану (поле **SREG**) і час виконання (поле **Stop Watch**). Причому ці значення можна змінювати в процесі відладки. Наприклад,

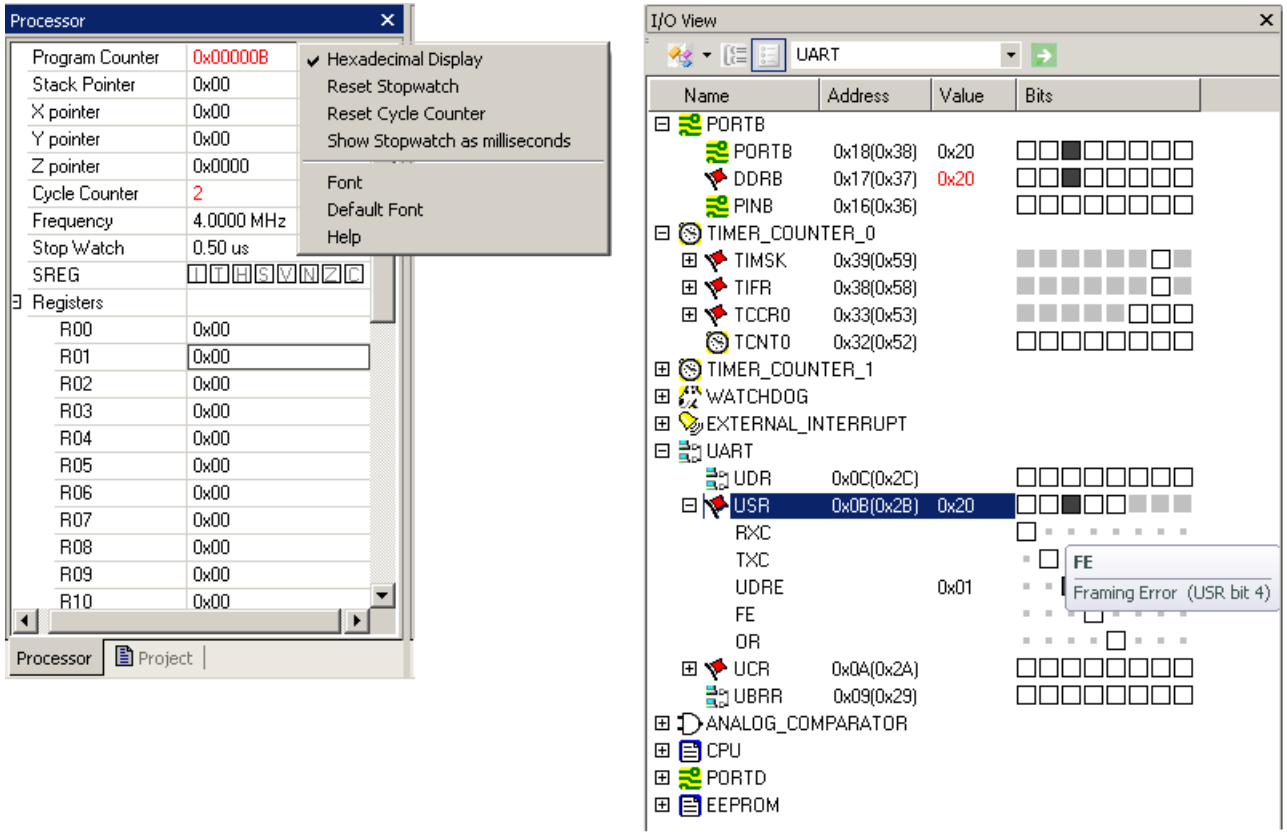
щоб визначити час або кількість тактів, за яку виконується певний фрагмент програми, потрібно спочатку клацнути правою клавішею мишки у вікні **Processor** і в спливаючому вікні вибрати пункт **Reset Cycle Counter** (**Рис. 9, а**), що обнулить значення полів **Cycle Counter** та **Stop Watch**. Після виконання фрагменту програми в рядку **Cycle Counter** буде знаходитися кількість тактів, а в рядку **Stop Watch** час виконання даного фрагменту.

У вікні **Processor** також можна переглядати та змінювати при потребі вміст 32 регістрів загального призначення (**Рис. 9, а**), які знаходяться в рядках **Registers** (регістри R0-R31).

У вікні **I/O View** зібрані регістри всіх периферійних блоків МК, а також регістри стану SREG та управління МК MCUCR. Регістри вводу-виводу згруповані по функціональним блокам і також доступні не тільки для перегляду, але і для редагування (**Рис. 9, б**). Наприклад, щоб налаштувати вивід PB5 як вихід з одиничним станом, потрібно клацнути на розряді 5 напроти регістра DDRB та на розряді 5 напроти регістра PORTB. Навівши курсор на певний біт регістра також можна отримати контекстну підказку про призначення цього біту (**Рис. 9, б**).

Для покрокового виконання програми можна скористатися клавішами **Debug/Step Into** (F11) або **Debug/Step Over** (F10). Різниця між ними полягає в тому, що F11 при виклику підпрограми здійснює перехід всередину тіла підпрограми, а при F10 підпрограма лише виконується і відбувається перехід до наступної команди. Якщо потрібно перескочити якийсь фрагмент програми, наприклад цикл, можна скористатися командою **Debug/Run to Cursor** (Ctrl+F10). Для цього потрібно розмістити курсор в тому рядку, на який слід перейти і натиснути Ctrl+F10 або вибрати цю команду через меню.

Для автоматичного виконання програми потрібно вибрати команду меню **Debug/Run** (F5). Для виходу з режиму автоматичного виконання програми та переходу в покроковий режим потрібно вибрати команду меню **Debug/Break** (Ctrl+F5). Для скидання МК в процесі виконання програми (формування сигналу RESET) потрібно вибрати команду меню **Debug/Reset** (Shift+F5).

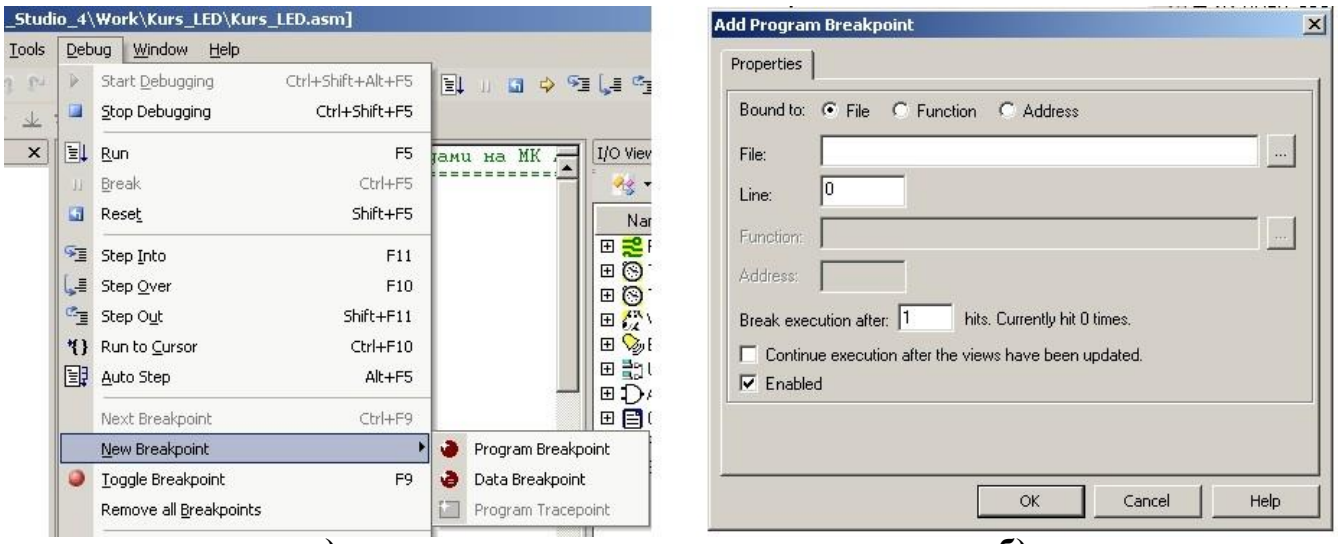


а) **б)**

Рис. 9. Вікна відображення поточного стану ядра МК (а) та периферійних одулів (б)

AVR Studio дає змогу встановлювати точки зупинки. Зупинка може відбуватися при досягненні певної адреси (програмна зупинка) або настанні певної події. Точка зупинк за адресою встановлюється напроти певної команди, при досягненні якої автоматичне виконання програми зупиниться і програма перейде в режим покрокової відладки. Щоб встановити точку зупинки на певній команді потрібно розташувати в рядку з командою курсор і вибрати пункт меню **Debug/Toggle Breakpoint** (**Рис. 10, а**). В рядку напроти вибраної команди появиться червоний кружок. Тепер якщо при відладці включити автоматичний режим (командою **Run**), відладчик при досягненні точки зупинки зупиниться і буде очікувати наступних дій.

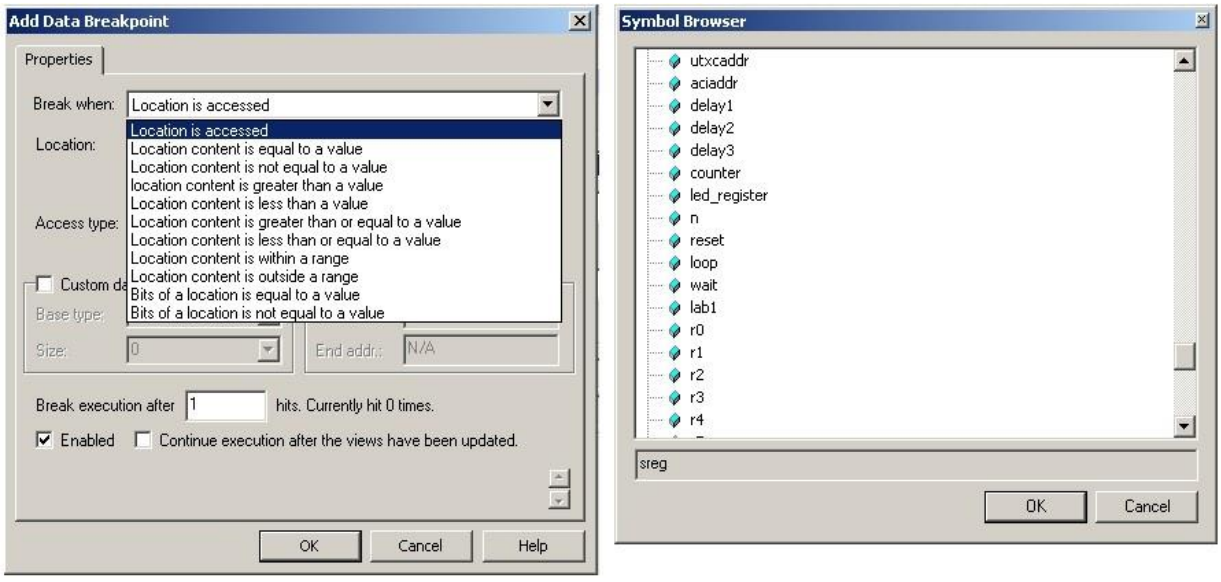
Також програмну зупинку можна встановити, вибравши пункт меню **Debug/New Breakpoint/Program Breakpoint** (**Рис. 10, а**). У вікні **Add Program Breakpoint** можна задати назву файлу, адресу чи назву функції, при досягненні яких відбудеться зупинка (**Рис. 10, б**).



а) **б)**

Рис. 10. Встановлення точки зупинки (а) та вікно параметрів програмної точки зупинки (б)

Для встановлення зупинки при досягненні регістром МК певного значення потрібно вибрати пункт меню **Debug/New Breakpoint/Data Breakpoint** (**Рис. 10, а**). У вікні **Add Data Breakpoint** можна задати подію, при якій відбудеться зупинка (поле **Break when:**), а також регістр який буде причиною зупинки (поле **Location:**, вікно **Symbol Browser**) (**Рис. 11**).



а) **б)**

Рис. 11. Вибір умови (а) та джерела зупинки (б)

Під час відладки можна переглядати та змінювати вміст регістрів загального призначення, оперативної пам’яті, регістрів периферійних пристроїв, пам’яті програм та енергонезалежної пам’яті даних, використовуючи

команди меню **View/Memory**, **View/Memory 2** або **View/Memory 3.** При цьому з'являється вікно **Memory** (**рис. 12**), де зі списку можна вибрати потрібну для перегляду ділянку пам'яті (в стовпцях виділених синім кольором відображається початкова адреса, далі йдуть дані, що знаходяться за цими адресами в шістнадцятковому та символьному представлені).

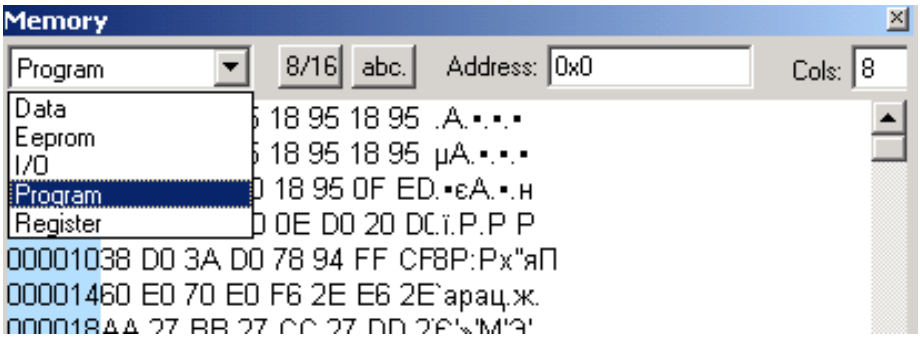


Рис. 12. Вибір області пам'яті для перегляду у вікні Memory

За значеннями змінних також можна спостерігати, вибравши команду меню **View/Watch**. При цьому появиться вікно **Watch.** У ньому потрібно клацнути правою клавішею мишки і у меню, що з'явиться, вибрати пункт **Add Item** (**Рис. 13**).

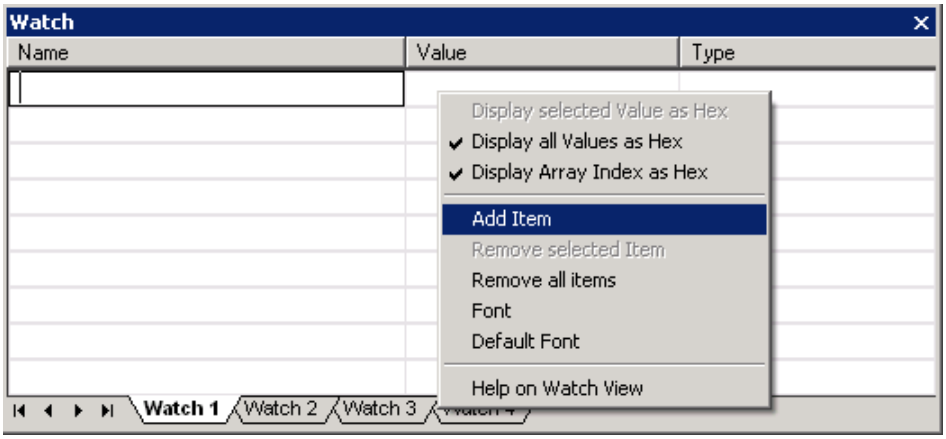


Рис. 13. Додавання змінних для спостереження в процесі відладки

Курсор автоматично переміститься в клітинку стовпця **Name** вікна **Watch,** в якому потрібно набрати ім'я змінної, за якою буде вестися спостереження. Наприклад

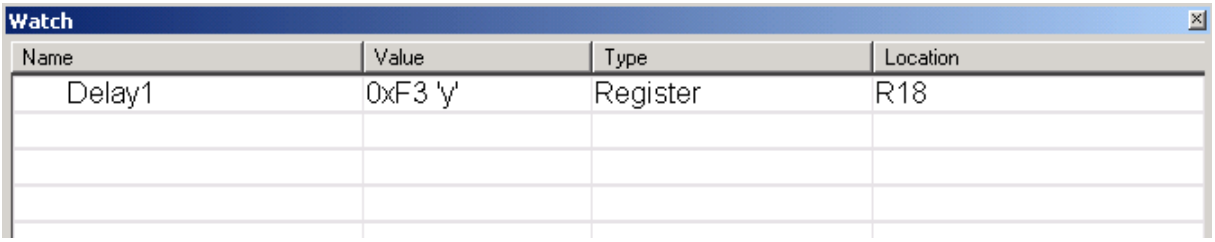


Рис. 14. Вид вікна Watch в процесі відладки

Тепер поточне значення змінної Delay1 буде відображатися в стовпці **Value** вікна **Watch** (**Рис. 14**). Дані можна відображати як в шістнадцятирічній, так і десятковій системах числення, встановлюючи або скидаючи прапорець **Display all Values as Hex** (**Рис. 13**).

Вийти з режиму відладки можна за допомогою команди меню **Debug/Stop Debugging** (Ctrl+Alt+F5). Відладка дає змогу переконатися в синтаксичній правильності створеної програми для МК, конфігурації периферійних блоків та взаємодії їх з підпрограмами і головною програмою.

2. ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

1. Вивчити теоретичний матеріал.
2. Вивчити основні властивості МК, необхідні для виконання лабораторної роботи.
3. Створити проект в AVR Studio, ввести програму, що наведена нижче, провести її асемблювання з генерацією файлу лістингу.
4. Запустити симулятор, в покроковому режимі виконати команди програми, при цьому:

- дослідити вміст вказівника стеку (відкрити вкладку CPU і виділити пункт меню Stack Pointer);
- дослідити вміст область пам’яті даних, в якій розміщено стек (вибрати пункт Memory в меню View, далі вибрати з випадаючого списку пункт Data і за допомогою смуги прокрутки перейти до найстарших адрес);
- дослідити зміни вмісту регістрів МК r16, r17, SREG (прапорці C та H) за допомогою вікна Processor в лівій частині екрану AVR Studio.

5. Зробити висновки по роботі.

Програма для введення в проект AVR Studio:

```
.include "m8def.inc"
        .org 0
        Ldi r16,low(RAMEND)
        Ldi r17,high(RAMEND)
        out SPH, r17
        out SPL, r16
main:
        ldi r16,10
        rcall subprog
        nop
m1:     rjmp m1

subprog:
        push r16
        in r17, SREG
        push r17
        ldi r17, 250
        add r16, r17
        pop r17
        out SREG, r17
        pop r16
        ret
```

Програма робить наступне:

1. Ініціалізація регістру вказівника стеку адресою найстаршої комірки пам’яті даних;
2. Запис в регістр r16 числового значення;
3. Виклик підпрограми subprog, в якій здійснюється:
   - розміщення в стеку вмісту регістру r16;
   - розміщення в стеку вмісту регістру статусу SREG через регістр r17;
   - встановлення прапорців C та H в регістрі статусу;
   - зміна вмісту регістрів r16 та r17;
   - зчитування вмісту регістру статусу SREG із стеку;
   - зчитування вмісту регістру r16 із стеку;
4. Вихід з підпрограми;
5. Зациклювання основної програми.

3. ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Лістинг програми, одержаної в **AVR Studio.**
3. Висновки.

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яке призначення програми **AVR Studio**?
2. Як здійснюється відладка в **AVR Studio** ?
3. Що таке точка зупинки ? Які ви знаєте типи точок зупинки ?
4. Які прапорці містить регістр статусу SREG ?
5. Яке призначення покажчика стеку?
6. Який механізм роботи стеку?

ЛАБОРАТОРНА РОБОТА № 4

Дослідження арифметичних та логічних операцій мікроконтролерів AVR

Мета роботи: надбання навичок програмування з використанням арифметичних та логічних операцій мікроконтролерів AVR фірми Atmel.

1. Теоретичні відомості

Арифметичні та логічні операції

Команди даної групи дозволяють виконувати такі операції над 8-бітними цілими двійковими числами: додавання, додавання з урахуванням переносу, десяткову корекцію, інкремент і декремент, віднімання, диз'юнкція, кон’юнкція, виключне або, інверсія, скидання, зсув. Опис команд наведений у табл. 1.

Група команд арифметико-логічних операцій Таблиця 1.

Мнемо-ніка	Операнди	Опис	Операція	Прапорці	Цикли
ADD	Rd,Rr	Додавання без переносу	Rd = Rd + Rr	Z,C,N,V,H, S	1
ADC	Rd,Rr	Додавання з переносом	Rd = Rd + Rr + C	Z,C,N,V,H, S	1
SUB	Rd,Rr	Віднімання без переносу	Rd = Rd - Rr	Z,C,N,V,H, S	1
SUBI	Rd,K8	Віднімання константи	Rd = Rd - K8	Z,C,N,V,H, S	1
SBC	Rd,Rr	Віднімання з переносом	Rd = Rd - Rr - C	Z,C,N,V,H, S	1
SBCI	Rd,K8	Віднімання константи з переносом	Rd = Rd - K8 - C	Z,C,N,V,H, S	1
AND	Rd,Rr	Логічне І	Rd = Rd · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логічне І з константою	Rd = Rd · K8	Z,N,V,S	1

OR	Rd,Rr	Логічне АБО	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Логічне АБО з константою	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Виключає АБО	Rd = Rd EOR Rr	Z,N,V,S	1
COM	Rd	Інверсія	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	Rd	Зміна знака (дод. код)	Rd = $00 - Rd	Z,C,N,V,H, S	1
SBR	Rd,K8	Установити біт (біти) у регістрі	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Скинути біт (біти) у регістрі	Rd=Rd · ($FF - K8)	Z,C,N,V,S	1
INC	Rd	Інкремент значення регістра	Rd = Rd + 1	Z,N,V,S	1
DEC	Rd	Декремент значення регістра	Rd = Rd -1	Z,N,V,S	1
TST	Rd	Перевірка на нуль або заперечність	Rd = Rd · Rd	Z,C,N,V,S	1
CLR	Rd	Очистити регістр	Rd = 0	Z,C,N,V,S	1
SER	Rd	Установити регістр	Rd = $FF	None	1
ADIW	Rdl,K6	Скласти константу і слово	Rdh:Rdl=Rdh:Rdl +K6	Z,C,N,V,S	2
SBIW	Rdl,K6	Відняти константу зі слова	Rdh:Rdl=Rdh:Rdl -K 6	Z,C,N,V,S	2

Команди множення присутні тільки у сімействі мікроконтролерів “Mega”. У мікроконтролерах сімейства ”Tiny” відсутні як команди множення, так і команди ділення. Розглянемо алгоритми множення та ділення цілих беззнакових чисел у мікроконтролерах AVR, у яких команди для виконання цих операцій відсутні.

Алгоритм множення беззнакових восьмирозрядних цілих чисел представлений на рис.1.

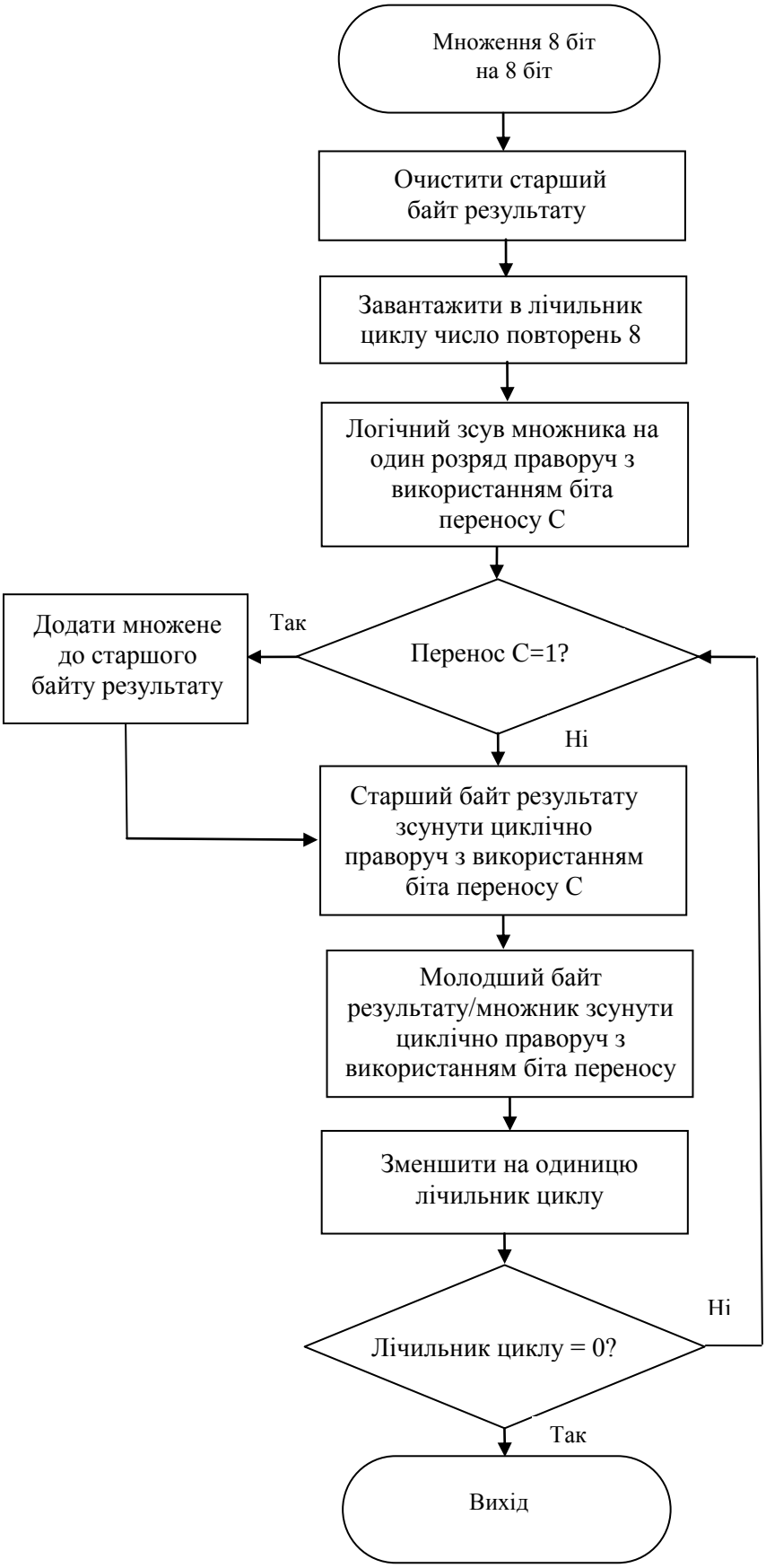


Рис. 1. Алгоритм множення 8-ми розрядних цілих беззнакових чисел

Алгоритм ділення цілого 16-розрядного беззнакового числа на восьмирозрядне представлений на рис. 2.

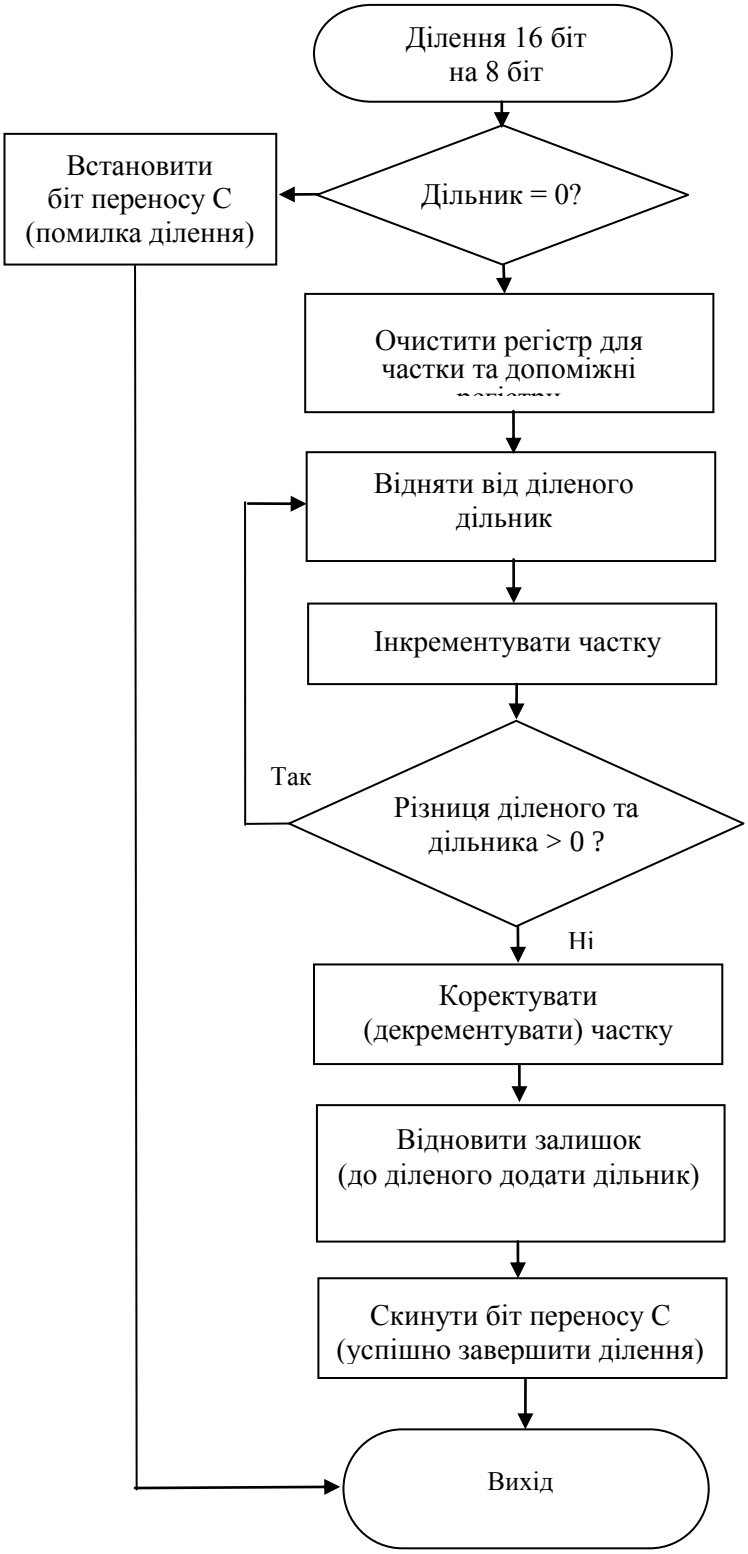


Рис. 2. Алгоритм ділення цілого 16-розрядного беззнакового числа на восьмирозрядне

2. Завдання до лабораторної роботи

6. Вивчити групу команд арифметико-логічних операцій мікроконтролерів AVR (табл. 1).

7. Скласти програми на мові Асемблер для реалізації алгоритмів множення та ділення беззнакових цілих чисел (рис. 1 та рис. 2).

8. Створити проект в AVR Studio, ввести складені програми та провести їх асемблювання з генерацією файлу лістингу.

9. Запустити симулятор AVR Studio та виконати програми. Праналізувати результати виконання програм.

10. Зробити висновки по роботі.

3. Зміст звіту

1. Список арифметико-логічних команд мікроконтролерів AVR.
2. Завдання до лабораторної роботи.
3. Тексти програм з коментаріями.
4. Результати роботи програм в середовищі AVR Studio.

4. Контрольні запитання

1. Перерахуйте команди арифметичних операцій мікроконтролерів AVR.
2. Перерахуйте команди логічних операції мікроконтролерів AVR.
3. Поясніть вплив команд арифметичних та логічних операцій на прапорці.
4. Поясніть алгоритм множення цілих беззнакових чисел.
5. Поясніть алгоритм ділення цілих беззнакових чисел.
6. Поясніть призначення логічних команд для операції «маскування» бітів.

ЛАБОРАТОРНА РОБОТА № 5

Робота з портами введення-виведення мікроконтролерів AVR

Мета роботи: надбання навичок програмування паралельних портів введення-виведення мікроконтролерів AVR фірми Atmel.

1. Теоретичні відомості

Мікроконтролери (МК) AVR оснащені 8-розрядними двонаправленими портами, причому кожна лінія порту може працювати як вхід або як вихід незалежно від інших ліній. Різні моделі 8-розрядних мікроконтролерів AVR мають різну кількість портів, але усі вони мають єдину архітектуру. Спрощена схема окремої лінії порту зображена на рис. 1.

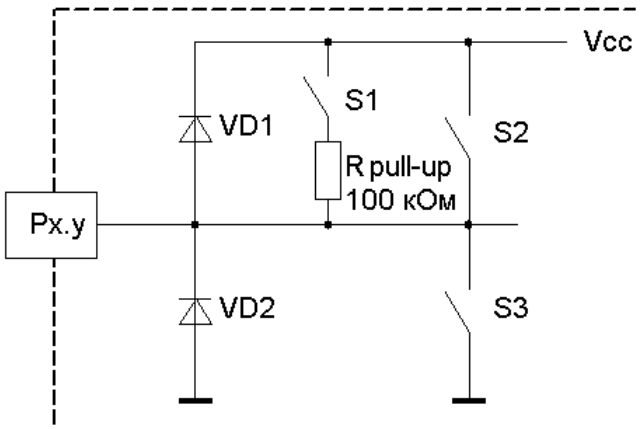


Рис. 1. Схема введення-виведення інформації через окрему лінію порту мікроконтролерів AVR

Діоди VD1, VD2 захищають електричну схему лінії введення-виведення від перевищення напруги. Якщо напруга на вході перевищує напругу живлення VCC, відкриється діод VD1 і лінію буде замкнуто на шину живлення. Якщо ж на вході буде напруга від'ємної полярності (тобто потенціал лінії буде менший за потенціал землі), відкриється діод VD2 і лінію буде замкнуто на землю.

Кожна лінія введення-виведення має *резистор підтяжки* RPULL-UP, який

"підтягує" потенціал лінії до рівня VCC. Його опір дорівнює 100 кОм. Резистор підтяжки можна програмно підключати або відключати, на схемі цю функцію виконує перемикач S1. Перемикачем S2 в режимі виведення на лінії встановлюється рівень логічної одиниці, а перемикачем S3 – рівень логічного нуля. Функцію перемикачів виконують ключі на польових транзисторах. Керування ключами здійснюється програмно встановленням або скиданням бітів у відповідних регістрах. Це робиться наступним чином.

Робота з усіма вузлами мікроконтролера (порти введення-виведення, інтерфейси, таймери, АЦП і т.п.) здійснюється через *регістри*. Кожен вузол МК має один або два (деколи більше) *регістрів керування*. Частина вузлів, що здійснюють приймання, передавання та обробку інформації, мають також *регістри даних*. Для того, щоб встановити режим роботи одного з вузлів або дати йому команду зробити певну дію, ви встановлюєте або скидаєте біти в регістрі керування, який за ним закріплений. Із свого боку вузол в процесі роботи повідомляє вас про зміну статусу (переповнення таймера, приймання байту інформації, закінчення перетворення і т. д.) також за допомогою бітів в регістрі керування.

Роботу з кожним портом забезпечують три 8-розрядних регістри: DDRx, PORTx та PINx, де x – літера позначення порту (A, B, C... і т. д.). Наприклад, порт B обслуговують регістри DDRB, PORTB, PINB. Кожен біт в цих регістрах відповідає окремій лінії даного порту. Номер біту будемо позначати літерою *n*.

Біти регістру DDRx визначають напрямок передачі даних кожної окремої лінії. Якщо біт DDRxn в регістрі встановлено, лінія *n* працює як вихід, а якщо скинуто – працює як вхід.

Функції бітів регістру PORTx залежать від того, в якому режимі працює лінія порту – як вихід чи як вхід. Якщо лінія Pxn працює як вхід, встановлення біту PORTxn підключає внутрішній резистор підтяжки. Для відключення резистору підтяжки лінії Pxn біт PORTxn необхідно скинути в нуль. Якщо лінія порту записом відповідного біту в регістрі DDRx сконфігурована як вхід, а резистор підтяжки відключено, лінія знаходиться у високоімпедансному стані (Z-стан). Якщо лінія Pxn сконфігурована як вихід, біт PORTxn визначає логічний рівень, який

видає лінія. При встановленні біту PORTxn лінія видає логічну одиницю (замикається ключ S2), а при скиданні біту PORTxn лінія видає логічний нуль (замикається ключ S3). Після включення живлення та під час скидання усі лінії усіх портів знаходяться в Z-стані. Усі можливі стани окремої лінії введення-виведення узагальнені в таблиці 1. Всі резистори підтяжки усіх портів разом можна відключити скиданням біту PUD в регістрі SFIOR.

Програмування станів окремої лінії введення-виведення Таблиця 1

DDRxn	PORTx	Стан лінії введення-виведення
0	0	Z-стан, резистор підтяжки відключено
0	1	Режим вводу, резистор підтяжки включено
1	0	Видача логічного нуля
1	1	Видача логічної одиниці

Регістр PINx призначений для зчитування стану лінії порту незалежно від значення бітів регістру DDRx, тобто через регістр PINx можна контролювати як вхідні, так і вихідні дані. Єдиний підводний камінь – це затримка, яку вносить схема лінії введення-виведення при оновленні регістру PINx після того, як стан лінії змінюється. Затримка невелика – від 0,5 до 1,5 періодів тактового сигналу в залежності від моменту зміни стану лінії, але її необхідно враховувати, якщо стан лінії зчитується одразу після її встановлення. Тому між командою запису та зчитування повинна бути витримана пауза тривалістю в один період тактового сигналу. Більш детально цей момент висвітлений в технічному описі мікроконтролеру ATmega8.

Є ще одне важливе застосування регістру PINn. В більшості нових мікроконтролерів AVR запис в цей регістр логічної одиниці викликає переключення відповідної лінії порту в протилежний стан, що є дуже зручним при формуванні на виході МК імпульсних послідовностей. На жаль, у МК ATmega8 ця функція відсутня, а ось більш нові ATmega48/88/168 вже підтримують цю функцію.

Команди для роботи з портами введення-виведення. Тепер розглянемо, які команди використовуються для роботи з регістрами DDRx, PORTx та PINx. Ці

регістри відносяться до регістрів введення-виведення ("введення-виведення" означає не тільки порти, а усі інші периферійні пристрої МК), і для роботи з ними призначена певна група команд. Можна зчитувати або записувати дані одразу в усі лінії порту (операції з байтами), або ж працювати з окремими лініями. Для запису в регістр введення-виведення байту даних призначена команда OUT, а для зчитування байту – команда IN. Спочатку розглянемо виведення даних в порт. Тут є одна особливість: для того, щоб записати в регістр введення-виведення якесь числове значення (константу), це значення спочатку командою LDI треба занести в один з регістрів загального призначення r16..r31, а вже потім командою OUT значення переноситься в регістр введення-виведення. Ось фрагмент програми, яка виводить в порт B число 10:

```
ldi r16, 0xFF
out DDRB, r16
ldi r16, 10
out PORTB, r16
```

Команда **ldi r16,0xFF** заносить константу 0xFF в регістр r16. Потім вміст регістру r16 заноситься в регістр напрямку передачі даних порту B DDRB. Цим самим ми настроюємо усі лінії порту B на виведення, оскільки в усіх бітах регістру DDRB будуть одиниці. Після цього в той самий регістр r16 ми заносимо число 10, а потім командою out PORTB, r16 виводимо це число в порт. Таким чином, ***запис констант в регістри керування портами здійснюється через регістри загального призначення***.

Восьмирозрядні мікроконтролери AVR мають 32 регістри загального призначення r0..r31, які напряму з'єднані з арифметико-логічним пристроєм і через які виконуються усі арифметичні та логічні операції. Команди IN та OUT працюють з усіма регістрами від r0 до r31. А ось команда LDI працює лише із старшою половиною цих регістрів, тобто з регістрами r16..r31.

Якщо порт необхідно налаштувати на введення, наприклад, прийняти байт даних, це здійснюється так:

```
ldi r16, 0x00
out DDRB, r16
```

```
ldi r16, 0xFF
out PORTB,r16
 in r0,PINB
```

Спочатку ми занесли нулі в усі біти регістру DDRB, чим перевели порт в режим введення, а потім записали одиниці в усі біти регістру PORTB, чим підключили до усіх ліній цього порту внутрішні резистори підтяжки. Після налаштування порту на введення даних зчитуємо стан ліній порту з регістру PINB в регістр r0.

Якщо налаштування порту здійснюється одразу після старту МК, записувати в регістри керування портами нульові значення необов'язково, оскільки після включення живлення всі регістри керування портами обнулюються.

Для роботи з окремими лініями портів використовуються команди SBI та CBI. Цими командами примусово встановлюється або скидається зазначений біт в регістрі введення-виведення, в тому числі і в регістрах керування портами.

2. Завдання до лабораторної роботи

11. Ознайомитися зі схемою ліній паралельних портів введення-виведення (рис. 1); з регістрами, які забезпечують роботу портів; а також з командами для роботи з портами введення-виведення.

12. Ознайомитися з принциповою електричною схемою для виконання лабораторної роботи (рис. 2).

13. Скласти програму на мові Асемблер, при виконанні якої мікроконтролер повинен виконувати функції шифратора: при натисканні однієї з 8 кнопок SW1÷SW8, підключених до порту D, в порт C (PC0÷PC3) повинен виводитись номер натиснутої кнопки в двійковому коді (рис. 2).

14. Створити проект в AVR Studio, ввести складену програму та провести їх асемблювання з генерацією файлу лістингу.

15. Запустити симулятор AVR Studio та виконати програму. Праналізувати результати виконання програми.

16. Зробити висновки по роботі.

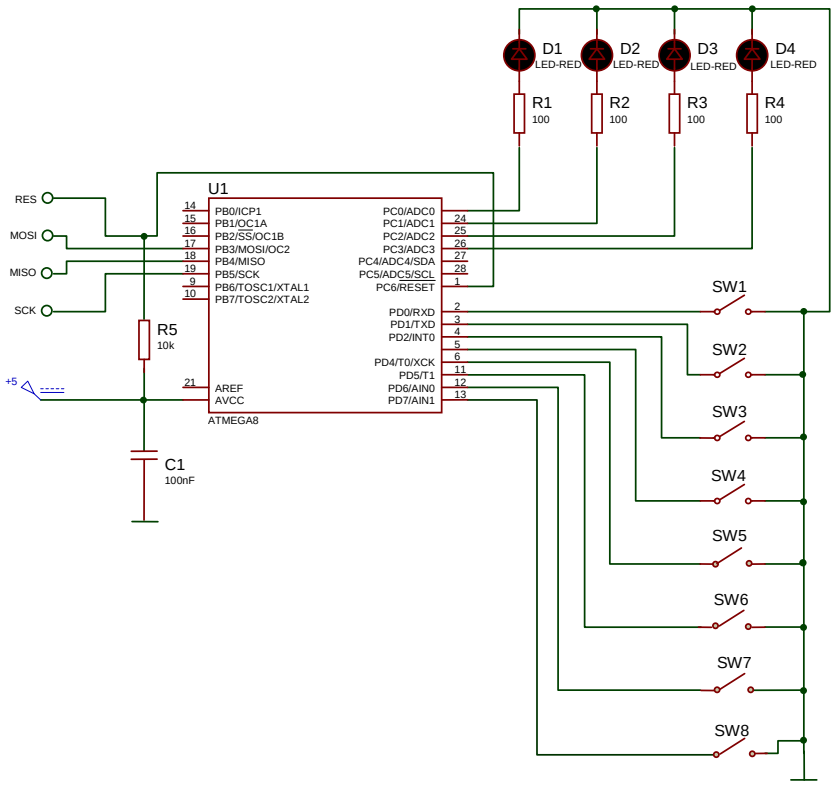


Рис. 2. Принципова електрична схема для виконання лабораторної роботи

3. Зміст звіту

1. Принципова електрична схема для виконання лабораторної роботи.
2. Завдання до лабораторної роботи.
3. Текст програми з коментаріями.
4. Результати роботи програм в середовищі AVR Studio.

4. Контрольні запитання

7. Наведіть спрощену схему введення-виведення інформації через окрему лінію порту мікроконтролерів AVR.
8. Яке призначення резисторів підтяжки портів введення-виведення?
9. Які регістри забезпечують роботу портів введення-виведення?
10. Охарактеризуйте функції бітів регістрів DDRx, PORTx та PINx.
11. Які існують команди для роботи з портами введення-виведення?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Михайлов С.Р. Мікропроцесорна техніка. Однокристальні мікроконтролери: навч. посібн. [для студ. вищ. навч. закл.] – К.: Кафедра, 2014. – 124 с.
2. Схемотехніка електронних систем: у 3 кн. Кн. 3. Мікропроцесори та мікроконтролери: підручник [для студ. вищ. навч. закл.] / [В.І. Бойко, А.М. Гуржій, В.Я. Жуйков та ін.] – [2-ге вид.]. - К.: Вища шк., 2004.- 399 с.
3. Сташин В.В. Проектирование цифровых устройств на однокристальных микроконтроллерах / В.В Сташин, А.В Урусов., Мологонцева О.Ф. - М.: Энергоатомиздат, 1990. - 221 с.
4. Бродин В.Б. Микроконтроллеры. Архитектура, программирование, интерфейс / В.Б. Бродин, И.И. Шагурин - М., ЭКОМ, 1999.- 400 с.
5. Бродин В.Б. Системы на микроконтроллерах и БИС программируемой логики / В.Б. Бродин, А.В. Калинин – М.: ЭКОМ, 2002. – 400 с.
6. Белов А.В. Самоучитель по микропроцессорной технике / Белов А.В. – СПб.: Наука и Техника, 2003. – 224 с.
7. Белов А.В. Конструирование устройств на микроконтроллерах / Белов А.В. – СПб.: Наука и Техника, 2005. – 256 с.
8. Белов А.В. Создаем устройства на микроконтроллерах / Белов А.В. – СПб.: Наука и Техника, 2007. – 304 с.
9. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR / Белов А.В. – СПб.: Наука и Техника, 2008. – 544 с.
10. Каспер Э. Программирование на языке ассемблера для микроконтроллеров i8051 / Каспер Э. – М.: Горячая линия-Телеком, 2004. – 191 с.
11. Хартов В.Я. Микроконтроллеры AVR. Практикум. Издательство МВТУ им. Н.Э. Баумана. 2012. - 280 с.
12. Белов А.В. Разработка устройств на микроконтроллерах AVR. СПб. Наука и Техника. 2013. – 528 с.